uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2017-2018

Trabajo Fin de Grado

"Aprendizaje por refuerzo dentro del sistema de combate de un juego de rol táctico"

Alberto García Arroba Parrilla

Tutor

Daniel Borrajo Millán

Leganés, 2018

ABSTRACT

The growth experienced in technology has opened tons of new fields of study. Some of them has stood out especially in the last years like the cases referring to Artificial Intelligence and Videogame Development. Not only their individual evolution in the market but the strong connection stablished between them, have been settled in our modern society and, what is more important, they have both been standardized.

The developed project aims to create a videogame with several gaming mechanics which contains a Machine Learning process able to confront players in a combat environment. The purpose of the study lies in verifying the validation of an agent trained using Reinforcement Learning in a turn-based combat game. Moreover, this agent must be able to present a well-balanced confrontation in which the players should not beat it so easily but also players must be able to win.

Through implementation using Python and the construction of a complete and logic state of the agent, several trainings are executed so that the best result in every situation can be obtained. In addition, thanks to a pre-stablished architecture of the intelligent system, it allows future changes to append new functionalities to make the game more appealing.

Key words: Artificial Intelligence, Intelligent System, (Computer) Architecture, Machine Learning, Python, Agent

AIM OF THE PROYECT

In order to understand the aim of the project, a few aspects must be explained. The process followed in this project is divided into two main parts.

The first part aims at creating an architecture of a Machine Learning agent capable of inducing knowledge in a combat scenario. This agent must be able to decide by itself which action will provide it the best results in every situation that can be encountered.

The second part, on the other hand, aims at developing a system in which the agent of the first part can interact with the players: a turn-based game. However, this game has a basic structure as the main objective of the project is to test this agent, and not testing the mechanics of the game. Mechanics of this game are divided in two: exploration through a map and combat against different types of enemies. Both parts are related as the enemies in the combat system act based on the agent's model developed in the first part.

But first, this agent must be trained so that it's able to execute proper actions. This process can't be made manually as it requires lots of training sessions, so a human agent is needed. To do so, preferentially, one or more players must play several rounds in a modified combat environment to register their actions in every situation. After that, training can be performed, and a model is obtained, which is used by the enemies to execute actions in the combat section of the game.

Knowing both parts of the project, the objective can be stated. The main objective of the project is divided into two connected parts: on the one hand, to study if a Machine Leaning model using the algorithm of Q-Learning in a turn-based game provides good results for the AI and, on the other hand, the creation of a software capable of performing every task previously mentioned that can be incorporated in an even bigger system which gives purpose to the agent, a videogame.

Besides, the first objective includes a sub-objective that has a secondary meaning in the project. The first one also aims at creating a combat environment well-balanced for both sides: players and non-playable characters. For this reason, victory for the players can't be a nearly impossible task to perform but also it can be too easy to obtain. It must be applied to cover the maximum number of players depending on their skill level and to make this encounter entertaining.

This concept is also focused on creating a more detailed and complete game in a future.

GAME MECHANICS

Knowing the mechanics of the videogame is key to understand the decisions taken in some sections of the project. For that reason, in this section, the mechanics and features of the game are explained.

First, as the game lacks a graphic representation of the progress, all useful information and interactions with the users are performed by console. Players only execute actions presented by the game by choosing one of the several options shown in the screen. Moreover, in case they are two or more, they must discuss group decisions.

The game starts asking how many users are going to play. This attribute can take values between 1 and 3, but only with two players the agent is implemented. After that, every player can choose one of the three different classes and the initial attributes points. Then, they choose the weapon and, altogether, the quest they want to do.

The progress along the game is simple: there is a map with nodes connected between them in a specific form that allows the players to move along them. Every time players reach a new node of the map, they enter a combat scenario in which they face the same number of enemies as them.

Every data of the game is accessed via files that store all features of the characters, enemies and environment in general. Therefore, the main mechanics of the game engine are reading and writing from files.

COMBAT

In the combat environment, both enemies and players do a base amount of damage that suffers modifications (rises and drops) based on different aspects that will be mentioned later.

Combat mechanics follow a common procedure as other games of the same genre or even board games based on turns. In this game we are dealing with 2 teams: players and non-playable characters (NPC); and each one of them can only execute one action on their turn.

Starting with players, every character alive executes the action they choose until all members of the team have done it then, it's the turn of the enemies, that follow the same procedure as the players: all alive enemies execute their action. In this particular case, enemies' action is limited to attack, while players can attack or consume a healing potion.

PLAYERS AND ENEMIES

Every player-controlled character has few features that can be chosen by them: class, attributes and weapon. Besides, each character has a health bar that when it reaches 0 points, the character dies.

The attributes represent different beneficial aspects of the character. These are 3: Strength, Resistance and Vitality. The first one, Strength, determines the additional damage a character is going to do; the second one, Resistance, determines the damage taken reduction and, the last one, Vitality, determines the maximum value of health points a character can have. Each attribute provides its corresponding benefit according to a specific formula.

The class determines how the player is going to play against specific types of enemies and stablishes the initial values of the attributes. This feature has 3 possible values: Bear School, Griffith School and Wolf School. Each class will start with a higher value of Resistance, Vitality and Strength, respectively. But, what is more important, its value affects directly the damage done to specific types of enemies.

On the other hand, the enemies only have one feature besides health points: its type. The enemies can be, according to the type: spectres, hybrids, humans, beasts, necrophages and insectoids. Each type has several subtypes of enemies that share the same characteristics but makes the combat less monotone.

The relation between class and type of enemy is what gives an interesting mechanic to the combat. Its functioning is simple: there are classes which do more damage to a specific type of enemies and do less damage to another type; and, from the other perspective, a type of enemy does less damage to some classes and more damage to another.

More specifically, Bear School characters do more damage to beasts and insectoids but less to hybrids; Griffith School characters do more damage to spectres and necrophages but less to insectoids; and Wolf School characters do more damage to hybrids and humans but less to spectres.

LOADOUT

Each character in the game has one weapon and a number of potions between 0 and 2, both included. Weapons don't wear out while potions are consumed on use.

Each weapon modifies the character's base damage and, additionally, provides some beneficial effects on the attacks. These effects can affect the additional damage done to a specific type of enemy or a rise in an attribute value.

Potions can be used only on combat and when it is their turn. It restores 5 points of health to the character who uses it.

MAP NODES

When combat is over in a map node, players can perform several actions that provide advantages in future encounters.

The most important feature of this situation is the chest that are randomly generated in the zone. There can be as many chests as players on the game. Every chest opened generates a random weapon from the pre-stablished arsenal list and can generate, at most, as many potions as players on the game.

Players have the chance to change their weapon with the one generated on the chest or simply leave it on the floor. Also, players who have used at least one potion, can take one of the potions generated on the chest.

After that, players can move to another node of the map, choosing the new node together.

MACHINE LEARNING

Among all different approaches in Machine Learning, the chosen one is Reinforcement Learning with the algorithm of Q-Learning. This algorithm needs some parameters fixed to proceed with a proper training.

As mentioned before, the model that is obtained from this algorithm works only for encounters against two players. If it was only one player, the agent wouldn't need to choose; and with 3 players, the state space would be too big. In future remodelling of the game, another algorithm can be stablished for these other cases.

REINFORCEMENT LEARNING AND Q-LEARNING

The structure of the Reinforcement Learning is composed of two main parts: the agent and the environment. They are both connected in both directions and each one has a specific task within the Learning.

The agent follows the model built and executes an action depending on the information it has, while the environment provides only a specific amount of information to the agent. The information given is divided in state and reward.

The state determines the situation in which the agent is performing, and it is marked by some attributes related to the problem. When the agent executes an action, the environment returns the new state and a reward based on how good that action was. If the action leads the agent to a fetching state, the reward is bigger than when the action leads to a bad state.

On the one hand, if every action in one state leads to bad results, the agent must be able to learn that those states should be avoided, while, on the other hand, if every action in one state leads to good results, the agent must be able to learn to prefer this state among others.

In the particular case of Q-Learning, the agent works with a table called Q Table with as many columns as different states and as many rows as possible actions. This table is filled in the training process according to the state the agent is in and the action taken.

The training process follows a repetitive path starting with the initialization of the Q table with zeros. For each session, the attributes of the state are randomly generated, aiming to reach as many states as possible. Unlike supervised and unsupervised learning, the training set is not fixed as one state can be visited more than once.

The purpose of this Learning is for the agent to know which action or actions must be executed in every state so that the accumulated value of reward is maximized. The path built from these actions is called policy. This policy is determined by two facets: first, the agent must analyse how good an action is in every state and, second, how the agent uses the information provided to execute the best actions.

EXPLORATION VS EXPLOITATION

One of the greatest problems in this method of learning is the separation between choosing exploration and exploitation.

Exploration consists on extending the visits to the greatest amount of states in the state space in order to find better and more promising solutions that haven't been refined or improved. This strategy implies diversification of the search so that the model doesn't stuck in a local maximum and tries to find a global optimum.

Exploitation strategy, on the other hand, reduces the search space to improve the favourable solutions that the model already has. The dedication is focused on the refinement and not in diversification, as exploration does.

Therefore, exploration is referred as global search while exploitation is referred as local search. Before starting the training sessions, a study must be made to extract which strategy is going to be more beneficial to the project at hand: boosting exploration, boosting exploitation or stablish a balance between both. Although including a bit of both

is necessary, because, without exploration, optimums can't be found and, without exploitation refinement can't be achieved to optimize results.

Even though these strategies present two different approaches, balance between them produces a beneficial effect in many domains within Reinforcement Learning. For that reason, each project should have balanced patterns between both strategies and a continuous control and management of them.

STRUCTURE OF THE AGENT

The most important structure to build in Q-Learning is the **state**. The attributes composing the state need to be relevant for the agent to perform the action as it represents the environment. These attributes must be composed by elements of the game that stablish the conditions of the game in every situation. The aim of this structure is letting the agent know certain information of both players to act based on the values of the attributes.

Therefore, the state is built as:

$$< R_{H1} , R_{H2} , C_{H1} , C_{H2} , HP_{H1} , HP_{H2} , HP_{NPC} , Type >$$

Being these attributes:

- **R_{H1} :** Resistance of Character 1
- **R_{H2} :** Resistance of Character 2
- **C_{H1} :** Class of Character 1
- **C_{H2} :** Class of Character 2
- **HP_{H1} :** Health points of Character 1
- **HP_{H2} :** Health points of Character 2
- **HP_{NPC} :** Health points of the agent (*NPC*)
- **Type :** Type of enemy

This state can be built using another combination of attributes with different values and using a different amount of them. However, in this project this construction is the only one used as it is considered complete. When the agent wants to decide who is it going to attack, it needs to measure the damage that it is going to do to every player. The Resistance of each player lets it know the reduction of the damage and the combination of class and type let it know if additional damage can be done to any of the players. Also, the health points are extremely important because killing a character provides a remarkable reward to the agent.

The values of the attributes are put into groups that share characteristics according to its values. For example, the health points are grouped based on their values in groups where the output of an attack is the same: one group has only the 0 which represent the character is dead, another group gathers all the values where the character will die regardless the damage taken and the other ones represent the amount of attacks needed to kill a character. These groups are made to reduce the state space but, as they are significative, it doesn't affect the results that much.

For each state, the **actions** that can be done are reduced to two: attack character 1 or attack character 2. The agent can still attack a dead character but will provide a bad reward for that reason.

Each agent will receive its own **reward** after players had executed their actions in the next turn. The reward is not obtained right after the agent executes its action because the agent needs to know the consequences of its performance based on consequent actions.

This reward depends on situation shown to the agent and the response to them. If the agent attacks a character, the accumulated reward increases as much as the damage done because it needs to learn that decreasing characters health points is a good action. If, in addition, it kills the character, the environment provides it with extra 100 points of reward. On the other hand, if, by any chance, the action chosen leads to the death of any of the agents, the agent receives a bad reward of -200, as it needs to learn that those actions have negative consequences.

HUMAN AGENT

Training in this kind of Learning must have, at least, as many sessions as possible states. However, it is advisable to revisit states to boost exploitation strategy. Training, therefore, needs to reach thousands of sessions to do so.

To train the system, a modified environment must be created only based on a combat encounter between the agents and players. But thousands of training sessions are used and doing it manually, using a real player, may be tedious and too slow. That's why a human agent is used to face these encounters against the agent.

The human agent must analyse the state it is in and return the action to take: attack enemy 1 or attack enemy 2. Thus, the construction of the human state must contain all related attributes relevant to the case, and they don't need to match with the ones used in the machine learning agent's state. The state, then, is formed by:

$< F, C, T_{NPC1}, T_{NPC2}, HP_{NPC1}, HP_{NPC2} >$

with:

- **F :** Strength of character
- **C :** Class of character
- **T_{NPC1} :** Type of enemy 1
- **T_{NPC2} :** Type of enemy 2
- **HP_{NPC1} :** Health points of enemy 1
- **HP_{NPC2} :** Health points of enemy 2

These attributes, like what happened with the machine learning agent's state, are grouped according to its values, always keeping a coherent and logic aggrupation.

Aiming to explore more possibilities and extend the number of tests to perform, two different human agents are built. The first human agent emulates actions based on two unexperienced players, therefore, actions can even be considered random. The second

agent, on the other hand, is created based on rules that determine which action is going to maximize the damage in every case scenario.

For the first agent all actions performed by the two players involved were recorded for several games and were stored in a file associating the action chosen with the corresponding state they were in. After those games, as not all states are visited, the missing ones are completed with random actions. Completion of these missing states doesn't suppose a big percentage of them as several states are visited in each game and generally, twice.

The second agent, unlike the first, covers all the state space from the beginning and doesn't require completion. The construction of states and actions is performed outside a game and is based on rules. These rules focus on maximizing the damage to be done, so elements like class of character, type of enemy and strength must be included. Theoretically, this agent is going to perform the best action in every situation.

The purpose of the first agent is to act like an unexperienced player while the second agent aims at emulating a more experienced player in this kind of games and, particularly, this game and its mechanics.

TRAINING

Training of the system is a crucial step to obtain a functional agent capable of executing actions based on its own knowledge and information extracted from the environment. To do so, several conditions must be stablished before beginning the training.

As the training is based on combat, these conditions must be defined in this type of environment. For that reason, a turn-based combat system is structured where the human agent faces the machine learning agent. With the goal of boosting exploration in the state space, each game starts with random values for every attribute defined in the state of the machine learning agent. This leads to varied durations of the games as it can start with some character or enemy whose health points are close to zero, having games of 1 turns or games with 6.

Besides the state attributes, Q-Learning algorithm has a few parameters whose values need to be stablished. These parameters affect directly the way the agent learns, and the approach taken before a quandary. These parameters are Alpha, Epsilon and Gamma.

Alpha, α, represents the learning rate and can take values between 0 and 1. This parameter determines if new information obtained about a state has a greater weight than the already existing information about it. It marks the 'aggressivity' when updating values of the Q table. When its value is close to 1, basically, the old value is replaced by the new one; while, when is close to 0, the old value of the Q table is slightly modified. Therefore, it is advisable to start training with Alpha close to 1 as, initially, Q table has no values; and, as training progress, reduce the value of Alpha reaching a value close to 0.

Epsilon, ε, is used when working with epsilon-greedy strategy. This strategy works directly with the approach to exploration and/or exploitation. At the moment the agent chooses which action is going to execute, a random number is generated. If the result is

smaller than Epsilon (with a value between 0 and 1), then a random action is executed. Else, the best action marked by Q table is executed. The value of Epsilon, therefore, is going to change throughout the training. It starts with a focus on exploration (higher value of Epsilon) and, as the agent learns, it is logic to reduce it to a value close to zero to settle into the strategy of exploitation of what the agent already knows.

Lastly, **Gamma**, γ, also called discount factor, multiplies the future rewards and affects the agent's choice of action. Gamma makes future rewards worth more than immediate rewards depending on its value. This value can be treated as a dynamic value that changes over the training sessions, but in this project, it keeps a constant value for each training.

The last feature to stablish is the number of **training sessions** for the agent to go through. Theoretically, the bigger the number of sessions, the bigger reach of the state space. To follow an evolution of the training sessions, these values are: 50, 100, 500, 1000, 2000, 5000, 10000 and 15000.

TESTS

To test the quality of all training several tests are prepared to cover all possible relevant cases. Each test is determined by a few characteristics: value of gamma, human agent involved, number of sessions and information provided to the agent.

Gamma receives 3 possible values: 0.1, 0.5 and 0.9. Depending on the training and the structure of the agent, Gamma can provide better results on different spectrums of its value. Generally, the bigger gamma is, the better results are obtained. However, in this project, the best results are obtained when gamma value is 0.5.

When training with the first human agent, the learning model obtains better results than when dealing with the second human agent; as this last one is a more difficult rival to play against for the machine learning agent.

Just as mentioned in theory, the bigger the number of training sessions, the bigger the percentage of victory against the human agents is.

The information provided to the agent is a detail that allows to study if an agent works better when it knows 100% of the relevant information of the problem at hand. More specifically, the information that is omitted to the agent is the weapon the player is wielding. On training, the agent knows that the human agent has a base damage of 4 but this value can change based on the weapon. Accordingly, if this information is not provided, the agent won't know exactly what the base damage its opponent is going to do and that can affect directly the outcome of some actions. After the tests, it can be noticed that when the agent knows 100% of information (there are no weapons), it performs better than when it doesn't know every inch of relevant information regarding the combat situation.

CONCLUSION

After finishing the study and development of this project all objectives previously established were successfully achieved.

Throughout the progression of the project, many changes were applied in order to converge into the established goal. Tests regarding training of the system suffered some necessary changes to reach as many cases of trained models as possible, but what suffered more changes was the game engine. The set up and establishment of the game mechanics was the easy part, while the balance of the game supposed more time invested. After the first trainings, agents couldn't beat a human player, not because it was badly trained but because the damage values of the combat were unbalanced and favoured players disproportionately. This, along with the formulas that affect the effects of the characters' attributes, suffered a great amount of changes.

The first objective was achieved thanks to procedures like the construction of a vast test set. However, to build a test set, a previous study of the algorithm was made to make that set coherent and avoiding redundancies. This eased a better attention to specific tests based on a certain value of parameters and the comprehension of the results obtained. For example, the evolution and possible values of the Q-Learning parameters.

The whole construction of a game engine with all associated elements, including the machine learning engine, mark the achievement of the second objective. As well as the first objective, a full study of certain aspects was necessary before the development of the system. This part may not need tests as the learning process does, but it needs the balance mentioned before as well as unitary tests that check the correct functionality of the system.

Not only the goals where achieved, but the game obtained as outcome turned out to be entertaining. However, many other aspects can be added to it so that it can represent a fully constructed game, for example, a graphic environment.

AGRADECIMIENTOS

Tras acabar no solo este proyecto sino los 4 años de carrera que vienen detrás, existen ciertas personas que han hecho más fácil este camino sin asfaltar y que deben recibir un cierto reconocimiento.

Para empezar, a aquellos que empezaron todo esto, mis padres, que estuvieron aguantando mis sandeces y descuidos a lo largo de esta etapa. También es importante agradecer que me enseñasen el verdadero significado de la independencia juvenil.

Al grupo de Cachopo, por sacarme de casa en los días que más lo necesitaba y por hacerme reír en los momentos más lúgubres. Y fundamentalmente por las conversaciones de carácter absurdo que definen nuestras quedadas.

Dentro de este grupo un miembro debe recibir un agradecimiento adicional, Pablo, que no solo me has tenido que soportar durante más de 3 años en la inmensidad de trabajos que hemos realizado, sino que también te tocaba aguantarme el resto del día. Será difícil encontrar un compañero más apropiado para mí.

A Dani, el señor de la luz, por su tolerancia a lo largo de estos años y por seguir ahí cuando las cosas estaban en sus peores momentos. Te debo mucho.

A Silvia, la salvadora, la guía que me redujo los años que iba a pasar en esta carrera, te agradezco toda la ayuda que me has ofrecido y la sabiduría compartida, no solo de la carrera sino de aspectos mucho más importantes de la vida.

Por último, pero no menos importante a Chuchi, que nos demostró que su con espíritu incansable y su determinación por una vida divertida puede llevarnos a lugares maravillosos. Nunca olvidaré todas las aventuras que vivimos en familia.

Tabla de contenido

Índice de tablas

Índice de figuras

1. INTRODUCCIÓN

Esta sección del documento se divide a su vez en 3 subsecciones: la primera se centra en la motivación del alumno a la hora de elegir el tema; la segunda debate el objetivo u objetivos que se quieren alcanzar con el desarrollo del estudio; y finalmente la estructuración de este documento.

1.1. MOTIVACIÓN

El mundo de los videojuegos ha evolucionado en gran medida desde que los titanes del género como *PacMan* (también conocido como 'Comecocos') o como el juego de *Space Invaders* apareciesen en los Arcades. Desde entonces hasta la actualidad, su crecimiento tanto en demanda como en oferta se ha disparado de forma exponencial convirtiendo a los videojuegos en un aspecto diario para muchas personas alrededor del mundo.

En sus inicios, la gran mayoría de estos videojuegos estaban centrados en que un jugador pusiese a prueba sus habilidades contra la máquina, y no enfrentándose a otros jugadores. Por ello, parte del equipo de desarrollo de estos géneros se han tenido que dedicar a crear una inteligencia propia para la máquina que juega contra los jugadores. En muchos casos incluso se ha llegado a crear una inteligencia distinta para cada elemento del juego, como es el caso de nuevo de *Pacman*. [1]

Con la creación de nuevas entregas y apariciones de juegos con modalidades distintas de jugabilidad, la presencia del modo Jugador contra Jugador (cuyo común acrónimo es *PvP* en inglés y JcJ en español) era mucho mayor. En este ámbito no es necesario estudiar y desarrollar una Inteligencia Artificial para controlar las acciones de la máquina, sino que se centran en otros aspectos del desarrollo del juego.

Sin embargo, hoy en día esto no ha supuesto el abandono de la Inteligencia Artificial en los comportamientos de los juegos. Como se ha mencionado antes, el mundo de los videojuegos ha crecido con un estallido de popularidad y ventas; y muchas empresas que los desarrollan no se han limitado a hacer juegos en los que solo participen los jugadores entre ellos. Muchos juegos tienen un enfoque dedicado 100% a la Inteligencia Artificial de los enemigos y otros muchos han decidido combinar ambas facetas para expandir las posibilidades.

La dedicación invertida en crear un comportamiento lo más cercano a la realidad para un elemento virtual de un videojuego, ya sea un enemigo, aliado o neutral, genera un mayor acercamiento a la situación que dicho juego quiere plantear. Por ello uno de los pilares más importantes en un juego de este estilo es la creación de una Inteligencia Artificial capaz de simular diferentes aspectos del mundo real.

El análisis de las distintas estrategias para implementar una Inteligencia Artificial supone una inversión elevada de tiempo y de recursos, pero a su vez da un refuerzo muy positivo al crear una inmersión tan dedicada a los detalles dentro del juego. Esta inmersión va a permitir un crecimiento en el interés de los jugadores que vayan a enfrentarse a esta

Inteligencia Artificial. No sólo la inmersión supone un acercamiento a la realidad, sino que el propio entrenamiento del sistema se asemeja mucho al mundo real.

1.2. OBJETIVO

Antes de entrar en detalle en los objetivos del proyecto es necesario exponer determinados aspectos relacionados.

El proceso seguido en este proyecto se separa en dos partes: la primera consiste en la creación de la arquitectura del sistema de aprendizaje donde se establecen las estructuras de los estados y las acciones disponibles junto con un entorno de entrenamiento automático; y la segunda parte se centra en la creación del esqueleto de un juego por turnos con el enfoque de Jugadores contra *NonPlayableCharacters*, es decir, personajes no jugables (también conocidos como *NPC*). Estas dos partes se relacionan directamente ya que los enemigos del juego seguirán el modelo entrenado de la primera parte.

Para ello se registrarán partidas de dos jugadores diferentes para obtener dos simuladores que permitan ejecutar un gran número de partidas de forma automática y más rápidas que si lo hiciese un jugador de forma manual. Tras el entrenamiento utilizando ese agente que simule a los jugadores, se podrá implementar la toma de decisiones de la IA (Inteligencia Artificial) dentro del motor del juego para las futuras partidas.

El principal objetivo del proyecto se divide en dos partes conectadas: por un lado, estudiar si el uso del Aprendizaje Automático con *Q-Learning* en un juego por turnos proporciona resultados favorables para la IA y, por otro lado, la creación de un *software* que sea capaz de realizar todas las tareas mencionadas previamente y que se pueda incorporar en un sistema aún mayor que dé un propósito al agente automático: un videojuego.

Dentro del primer objetivo se debe incluir que el entorno de combate por turnos sea favorable para ambas partes, es decir, que no esté desbalanceado para ningún lado y que el porcentaje de victorias no sea muy desproporcionado el uno del otro. Esto se debe aplicar para abarcar el máximo número de usuarios en función de su nivel de habilidad en juegos de este estilo.

Este concepto también está enfocado a la creación de un posible juego algo más detallado y completo para un futuro.

1.3. ESTRUCTURA DEL DOCUMENTO

A partir de este punto, el documento se divide en varias partes, estudiando en cada una aspectos relevantes para el proyecto. El orden en el que están estructuradas es importante para poder trabajar con una secuencia coherente de características.

El primer apartado, denominado Estado del Arte, se centra en la investigación previa al desarrollo del proyecto. Esta investigación está dedicada a estudiar cada uno de los aspectos relevantes del proyecto y permite elegir entre distintas alternativas o enfoques

aplicados a un aspecto en particular. Además, sirve para tener un conocimiento más amplio de alguno de los conceptos en los que está basado el proyecto.

El siguiente apartado del documento está focalizado en el análisis necesario antes de empezar con el desarrollo del proyecto. Aquí se estudia el alcance del trabajo, las posibles herramientas a utilizar y las funcionalidades. También se establecen las diversas acciones de los usuarios dentro del juego y se plantean las pruebas del sistema para comprobar su correcto funcionamiento.

Más tarde se estudia el diseño del sistema, empezando por el diseño del juego donde se explica qué mecánicas tiene y qué temática sigue. Se continúa revisando la arquitectura y la interfaz que el usuario ve cuando lo ejecuta y se acaba con la presentación de los distintos diagramas del software que representan los componentes, las clases y las secuencias dentro del sistema.

Se da paso al apartado de implementación y evaluación donde: por un lado, se explica cómo se ha desarrollado el juego junto con la estructuración del aprendizaje utilizado para entrenar al agente automático. Por otro lado, se hace una evaluación tanto de las pruebas establecidas previamente como de la valoración objetiva de usuarios de prueba.

Antes de acabar, se hace un repaso del entorno socio-económico donde se analizan los impactos en distintos ámbitos, el presupuesto y la planificación del proyecto.

Finalmente, se presenta una conclusión revisando los aspectos trabajados a lo largo del desarrollo del proyecto, analizando los objetivos establecidos y marcando rasgos que se puedan mejorar o añadir en futuros trabajos.

2. ESTADO DEL ARTE

Esta sección del documento está dedicada al estudio previo sobre aspectos relacionados con el tema del proyecto que puedan ayudar para el desarrollo del mismo, aunque algunos aspectos no guardan una relación directa.

Inicialmente se estudian los videojuegos como base para la temática en la que va a estar basado el juego, incluyendo el género de 'videojuegos por turnos' en el que está centrado y los 'juegos de rol' con los aspectos que han influido.

A continuación, se analiza el aprendizaje automático revisando sus aspectos más relevantes y los distintos tipos que existen, donde se dedica un apartado especial al tipo de aprendizaje utilizado en este proyecto. Además, se aludirá a la existencia de esta herramienta en los videojuegos y cómo se plantea en distintos escenarios.

Finalmente, se hace una revisión sobre artículos publicados sobre estos temas y proyectos similares con el fin de hacer una comparativa entre ellos y este proyecto.

2.1. VIDEOJUEGOS

Los videojuegos, o juegos electrónicos, son métodos de entretenimiento para una o más personas donde se utiliza un controlador (ya sea un mando, teclado o táctil entre otras herramientas) y un dispositivo que muestra imágenes para jugar. Todo videojuego tiene un objetivo o meta que el jugador o jugadores deben alcanzar y para ello se proporcionan distintas habilidades, herramientas o mecánicas. La variedad de lo que se proporciona es inmensa, teniendo casos muy sencillos como el mítico juego de *Pong* [2] como el de la figura 1, juego de *Atari* [3], donde el jugador sólo tenía una 'pala' para golpear una pelota y pasarla al campo contrario hasta que alguno de los dos jugadores fallase. Y luego casos más complejos, como por ejemplo *Age of Empires* [4] o juegos similares de estrategia, donde el jugador tiene la capacidad de construir una gran variedad de edificios, entrenar distintos tipos de soldados y puede utilizar un gran número de estrategias para ganar.

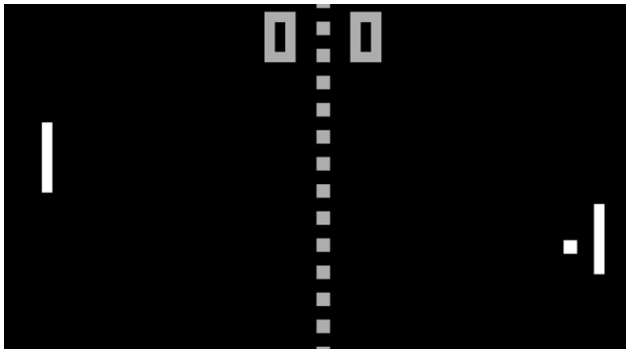

Figura 1: Juego de Pong

Dentro del mundo de los videojuegos, el abanico de géneros y temáticas se extiende hasta completar una lista de más de 50 elementos, teniendo géneros y subgéneros tan comunes como 'Disparos' o 'Acción'. Sin embargo, la división que importa en este estudio es aquella que separa a los videojuegos en 2 categorías: **Jugador Contra Máquina** o

Jugador Contra Jugador. La segunda categoría es indiferente para este proyecto, a diferencia de la primera que sí es relevante por la dedicación a un sistema inteligente capaz de enfrentarse a un jugador.

El desarrollo de una IA es un proceso que requiere un considerable tiempo de desarrollo y muchas pruebas para conseguir que sea la apropiada en cada ocasión ya que, dependiendo de su propósito, el enfoque varía. Todos los rasgos característicos del modelo que representa a la máquina definen la dificultad, la jugabilidad, la estrategia que hay que seguir para enfrentarse a ella y muchos más aspectos que estructuran la IA. Por ello, es imperativo hacer un estudio previo que permita a los desarrolladores y demás responsables crear un juego donde todas las características estén relacionadas formando una cohesión lógica. Si se omite, el proyecto puede perder coherencia y ocasionar dificultades a la hora de desarrollarlo.

Los videojuegos de Jugador contra la Máquina vienen en muchas formas y tamaños, pero dentro de este extenso grupo se debe prestar especial atención a un subgénero en particular: los videojuegos por turnos, también conocidos como videojuegos de rol táctico.

2.1.1. VIDEOJUEGOS POR TURNOS

El funcionamiento de los videojuegos por turnos se asemeja en gran medida a aquellos juegos de mesa que también se rigen por turnos. En los casos más generales, se separan dos bandos donde en cada uno de ellos hay al menos un participante. En el caso de los videojuegos, los jugadores se enfrentan a uno o varios enemigos marcando los dos bandos de esa forma. Dentro de estos bandos se debe seguir un orden determinado sin repeticiones por turno: primero realizan una acción cada uno de los jugadores en un cierto orden y luego los enemigos, en un orden específico, realizan una acción cada uno. Por lo que las acciones de cada elemento del juego (ya sea jugador o enemigo) solo se pueden ejecutar dentro de su turno, estando "bloqueadas" para el resto de los turnos. Esta secuencia se repite hasta que uno de los dos bandos alcanza el objetivo (generalmente acabar con el bando contrario) y se termina el juego.

Dentro de los juegos por turnos más influyentes en este proyecto se encuentra la saga de *Pokemon* y la saga *Final Fantasy* que, aunque no fueron los primeros en salir a la luz, tuvieron más repercusión que muchos de sus antecesores.

La jugabilidad de la saga *Pokemon* se separa en 2 partes fundamentales: exploración por un mundo y combate por turnos. La mecánica de combate por turnos se mantiene igual, ya sea 1 contra 1 o 2 contra 2, primero ataca un bando y luego ataca el otro sin interferir en el turno del resto. Los combatientes en ese juego se caracterizan por muchos aspectos, pero el más influyente para este proyecto es la cualidad que determina la tipología de *Pokemon* [5]. Dentro de los tipos tenemos ejemplos como: planta, fuego, veneno… y cada uno de estos tipos es fuerte contra un tipo, pero débil contra otro. Pongamos el ejemplo de los tipos agua, planta y fuego: como en el popular juego de piedra-papel-tijeras, el agua es fuerte contra el fuego y débil contra planta; el fuego es fuerte contra planta, pero débil contra agua y finalmente, planta es fuerte contra agua y débil contra fuego. La

existencia de estas relaciones de fuerza y debilidad en función del tipo de personaje genera una variedad adicional y hace el combate más interesante. Además del combate la exploración del mapa es otro punto fuerte de este juego ya que otorga al jugador una libertad a la hora de elegir qué camino tomar y qué riesgos correr.

Por otro lado, la saga de *Final Fantasy* se define como un videojuego de rol táctico donde el jugador controla un grupo de personajes y debe ir superando combates para desplazar a dichos personajes por un escenario. En este caso, a diferencia del anterior, se pueden manejar más de dos personajes aumentando la combinación de posibilidades dentro del combate. En cuanto a las acciones, se separan en ataques normales y especiales dando más variedad a la hora de atacar. Adicionalmente, también existe la faceta de exploración de un mapa dentro de este juego.

2.1.2. JUEGO DE ROL

Por mucha influencia que exista procedente de los videojuegos por turnos, los que más han influido tanto en videojuegos como en este proyecto han sido los juegos de rol clásicos. Antes incluso de la aparición de los *Arcades*, estos juegos sin relación digital ya tenían su popularidad entre gente de todas las edades por su fácil acceso y gran versatilidad.

Los juegos de rol de mesa se remontan a más de 50 años, aunque no como todo el mundo los conoce. Muchos de estos juegos de mesa simplemente incluían roles que eran interpretados por los jugadores y, más adelante en los años 60, muchos otros evolucionaron de las recreaciones históricas a la creación de historias ficticias o creativas. Dado que estas recreaciones se ambientaban con más frecuencia en la Edad Media, la temática principal era fantasía en esa época histórica. Estos juegos de mesa, aunque populares entre los jugadores, ocasionaron un impacto negativo en la sociedad y sobre todo en personas que no los jugaban por el número de crímenes asociados [6] [7]. Aun así, con la aparición de los videojuegos y la extensión a más usuarios, incluyendo gente más escéptica, este impacto negativo disminuyó notablemente.

Al no tener una computadora o consola que controle todas las acciones y reglas del juego, los juegos de rol recurren a otro recurso para tener todo medido y regulado: el máster. El papel de máster es el más importante del juego ya que abarca prácticamente todas las características del entorno y la regulación del juego. Al tratarse de algo creativo e inventado, sus responsabilidades abarcan desde la creación de una historia, mapa, enemigos y demás elementos de la partida hasta las acciones que los jugadores no efectúan, como, por ejemplo, las decisiones de los enemigos. El máster procura que todo sea coherente y organiza la narrativa asegurándose de que los jugadores siguen las reglas que ella o él haya establecido. Además de actuar de guía para los jugadores durante la partida, debe estar familiarizado con los documentos necesarios como el Manual de Jugador donde se establecen las reglas de creación de personajes y la jugabilidad desde el punto de vista del jugador. Finalmente, el máster no sólo debe lidiar con aspectos dentro del juego sino con elementos externos como enfrentamientos entre jugadores, capacidad para motivarlos e improvisar [8] [9].

Por otra parte, los jugadores solo se encargan de enfrentarse a las situaciones que el máster les presenta, desde una simple conversación con distintas aproximaciones hasta una pelea con enemigos. Por norma general, se utilizan dados para realizar acciones y dependiendo del resultado de la tirada, el resultado en el juego influye.

El progreso de la partida es simple: los jugadores empiezan en un mismo punto del mapa y el máster expone la historia y las opciones iniciales. A partir de ahí, los jugadores van moviéndose por el mapa previamente creado y resolviendo las situaciones que les presenta el máster hasta que alcanzan la meta o hasta que pierden la partida, es decir, cuando todos los jugadores tienen sus puntos de vida en cero.

Adicionalmente, los juegos de rol presentan más funcionalidades ya que lo planteado previamente haría al juego muy simple. A pesar de existir una gran cantidad de documentación, fórmulas y reglas, solo se expondrán aquellos aspectos más relevantes que se han llevado a los videojuegos actuales como lo son los atributos, experiencia, botín…

Los jugadores tienen una tarea inicial previa a que la partida empiece: la creación de sus personajes. Separando los rasgos físicos que no influyen en la jugabilidad, normalmente se elige la raza y la clase que, dependiendo de la temática, varían entre varias opciones distintas. En el caso de *Dungeons & Dragons* tenemos razas entre las que se encuentran centauros, elfos, enanos… [10] y clases como clérigo, bardo, druida, mago… [11] La elección de la raza y la clase va a afectar, entre otras cosas, a los atributos iniciales del personaje. Cada jugador tiene una combinación de atributos que influirán directamente en los resultados de las tiradas (de dados) de sus acciones. Por ejemplo, un atributo como el de fuerza repercute en el daño que haga el personaje mientras que un atributo de resistencia influye en la reducción de daño recibido; aunque también existen atributos que afectan a acciones fuera de combate como, por ejemplo, el carisma que da una modificación a los resultados cuando el personaje habla con un NPC o cuando intenta persuadir a alguien. Estos atributos van aumentando en función del equipamiento que lleven los personajes o al subir de nivel. Sin embargo, la lista de atributos depende de la temática y de cómo lo haya estructurado el máster.

Además de los atributos, las fichas de personajes tienen una sección dedicada al equipamiento que llevan que, dependiendo de la estructuración por parte del máster, puede referirse a armadura, arma, amuleto, anillo… Estas piezas de equipamiento proporcionan ventajas al jugador en ciertos aspectos y facilitan enfrentamientos futuros [12].

Es difícil que un juego de combates por turnos esté centrado puramente en la lucha ya que, a menos que el sistema de combate sea muy interesante, la jugabilidad se haría repetitiva y aburrida. Por ello se añaden más secuencias de acciones que el jugador pueda experimentar como, por ejemplo, escenas que contengan diálogos para contar una historia y/o, más comúnmente, un sistema de movimiento y exploración. Este último facilita el paso entre combate y combate y proporciona al usuario una sensación de avance y progreso, al igual que la experiencia y los niveles.

El sistema de experiencia representa lo bien que ha estado jugando un jugador recompensando con una cantidad de experiencia a todo aquel que complete un objetivo o

derrote enemigos. Todo jugador empieza en nivel 1 y cuando alcanza cierta cantidad de experiencia, sube de nivel y su progreso de experiencia vuelve a cero. Al subir de nivel, dependiendo de la estructuración del máster, los atributos del jugador aumentan de forma automática dependiendo de su raza y clase o el jugador puede elegir qué atributo o atributos mejorar.

El último aspecto relevante que queda por mencionar para este proyecto es la existencia de objetos favorables para los jugadores. Además del equipamiento, existen otros objetos que el máster puede añadir a la partida para ayudar a los jugadores a progresar y/o para aumentar la versatilidad del juego. Son objetos como, por ejemplo, pociones que restauran puntos de vida o pergaminos que aumenten uno o varios atributos por un turno. [13] Estos objetos se pueden obtener de cofres que aparecen en ciertas zonas del mapa o por el botín que sueltan los enemigos al ser derrotados. Dependiendo de la temática del juego, los objetos que se puede encontrar un jugador en estos cofres varían desde pociones hasta equipamiento pasando también por dinero virtual que tiene valor dentro de la partida.

Muchos videojuegos han utilizado estas cualidades que definen a los juegos de rol para caracterizar sus temáticas y/o jugabilidad y, por ende, es necesario mencionar dichos aspectos en este apartado ya que los juegos de rol son el promotor de la evolución de muchas entregas digitales.

2.2. APRENDIZAJE AUTOMÁTICO

El aprendizaje automático es el estudio de la informática que analiza el reconocimiento de patrones y análisis computacional en manos de las máquinas o, mejor dicho, de la Inteligencia Artificial. Esta rama de la informática provee a las computadoras de la capacidad para aprender por sí solas mediante el uso de casos de ejemplo o modelos. El principal objetivo es hacer que los ordenadores sean capaces de interactuar de forma autónoma mediante decisiones propias.

Para conseguir que una máquina sea capaz de ejercer tales funcionalidades se debe entrenar al sistema con un conjunto de ejemplos, también llamado conjunto de entrenamiento (en inglés *training set*). El conjunto de entrenamiento está compuesto de instancias de datos asociados al problema a resolver y cada instancia se define por una combinación de valores de los atributos que la componen.

El entrenamiento o aprendizaje se centra en extraer los patrones dentro de ese conjunto de instancias; a partir de ahí, mediante dichos patrones se realizan predicciones utilizando nuevos datos que se añadan al sistema.

Después de entrenar al sistema se realiza una validación para comprobar si se ha entrenado de forma adecuada al sistema, utilizando un conjunto diferente al de entrenamiento llamado conjunto de validación. Este apartado comprueba la calidad del entrenamiento previo. Finalmente, se realiza el mismo proceso con un conjunto de pruebas diferente a los dos conjuntos anteriores para probar el modelo entrenado. El conjunto de validación es opcional, pero es recomendable utilizarlo dado que permite

ajustar el modelo en función de los resultados con el conjunto de prueba y elegir el mejor para esos mismos resultados.

Por tanto, el conjunto total de datos se debe separar en dos si no se utiliza un conjunto de validación o en tres en caso de utilizarlo. El conjunto de entrenamiento debe ser superior al de prueba y para que este último conjunto sea válido debe cumplir dos condiciones: que sea lo suficientemente extenso como para producir resultados significativos y que el conjunto de entrenamiento tenga características similares a los datos futuros, es decir, que el conjunto de entrenamiento sea representativo. Un buen porcentaje de divisiones es 80% para entrenamiento y 20% para validación-prueba.

2.2.1. SUPERVISADO VS NO SUPERVISADO

Dentro de este concepto tan amplio que es el Aprendizaje Automático, tenemos varias divisiones que se diferencian entre sí por su estructura y enfoque. Por un lado, tenemos el aprendizaje supervisado frente al aprendizaje no supervisado, y, por otro lado, el aprendizaje por refuerzo. Aunque también existen otros menos utilizados como el aprendizaje semi supervisado y aprendizaje multi tarea.

En aprendizaje supervisado se trabaja con un conjunto de datos a los que se les atribuye una etiqueta llamada clase y donde, dados los atributos de entrada, se devuelve la clase a la que pertenece esa instancia. Esta clase representa una etiqueta que caracteriza a cada instancia. En funcionamiento, el algoritmo, tras el entrenamiento, obtiene un conjunto de instancias sin etiquetar (sin clase) y predice el valor de esa clase. Entre los problemas que se resuelven utilizando este tipo de aprendizaje se encuentran la clasificación y regresión con algoritmos como árboles de decisión, clasificación *Naïve Bayes* o regresión por mínimos cuadrados. Estas herramientas, más específicamente un clasificador, podrían utilizarse identificando a la clase como la acción tomada por el agente en el enfrentamiento del juego.

En caso de no tener una clase que etiquete a las instancias de entrenamiento, se recurre al aprendizaje no supervisado. Al no existir ninguna clase y solo poseer los datos de entrada, se estudia la estructura de los datos para encontrar un patrón o una organización. Se trabaja con problemas de *clustering* o agrupación y utiliza, por ejemplo, algoritmos de *clustering*.

Además de supervisado y no supervisado, tenemos otro tipo que se separa más de ellos al utilizar un enfoque distinto para el aprendizaje: el aprendizaje por refuerzo. A diferencia del aprendizaje supervisado y no supervisado, éste utiliza un sistema de recompensas o refuerzos para el proceso de entrenamiento.

2.2.2. APRENDIZAJE POR REFUERZO

El proceso detrás del aprendizaje por refuerzo es bastante claro y simple. Tenemos dos elementos cruciales que son capaces de comunicarse entre ellos: el entorno y el agente. Llamamos entorno a las condiciones externas que caracterizan el ámbito en la que actúa

el agente y llamamos agente a la máquina en la que está centrada el proceso de aprendizaje que va a efectuar las acciones.

El aprendizaje se rige por el **Proceso de Decisión de Markov** que se compone en cada momento de: un estado S y un conjunto finito de acciones A. Al tratarse de un avance temporal discreto, se deben tratar tanto el estado como las acciones en función de ese momento, $\mathbf{s_t}$ **y** $\mathbf{a_t}$. El proceso se compone de un bucle que se repite con las siguientes acciones también mostradas en la figura 2:

- El agente extrae un conjunto finito de estados S y las posibles acciones que se pueden ejecutar, también de carácter finito.
- Al ejecutar una acción, cuya elección se explicará más adelante, se obtiene un nuevo estado $\mathbf{s_{t+1} = a_t (s_t)}$
- Al haber efectuado esa acción, el entorno otorga un refuerzo $\mathbf{r (s_t, a_t)}$ cuyo valor es mayor cuanto más grande sea el beneficio.

Es importante aclarar que el agente desconoce las consecuencias cuando toma una acción específica en un estado determinado ya que tanto el estado siguiente como la recompensa no necesariamente tienen que ser conocidos a priori por el agente y varían en función del estado en el que se encuentre y la acción efectuada.

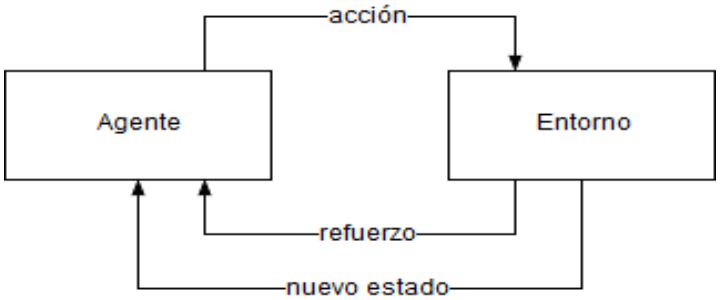

Figura 2: Elementos de Aprendizaje por Refuerzo

El propósito del aprendizaje por refuerzo reside en obtener qué acción o acciones hay que elegir en cada uno de los estados para que devuelva el máximo refuerzo y alcanzar el estado final. Se busca que el agente siga una determinada estrategia o política que devuelva qué acción ejecutar en cada estado y defina sus comportamientos. Esta política se rige por dos facetas: por un lado, el agente debe analizar cómo de beneficiosa es una acción en ese estado y, por otro lado, hay que estudiar de qué forma usa el agente lo que se le ha informado para elegir las mejores acciones.

La implementación más conocida y utilizada para el aprendizaje por refuerzo es ***Q-Learning*** en la que el agente trabaja con tuplas (estado, acción). Cuando un agente toma una acción en un determinado estado, el refuerzo que recibe de forma inmediata es relevante. Sin embargo, también son de interés los refuerzos futuros que se reciben en estados siguientes dentro de la política establecida. Para cada tupla (estado-acción) se asocia un valor Q que representa la acumulación de todos estos refuerzos. Aun así, el peso de los refuerzos inmediatos y el de los refuerzos futuros no necesariamente tienen que ser los mismos, por lo que se utiliza un factor de descuento (Gamma) cuyo valor está

comprendido entre 0 y 1 de tal forma que cuanto más grande sea gamma, las recompensas futuras influyen más. Esto se denomina refuerzo acumulado con descuento.

Para poder tener un valor de Q asociado a cada tupla, se debe obtener una fórmula matemática que devuelva ese valor en función de los elementos involucrados, pero, antes de ello, se ha de tener en cuenta ciertas consideraciones. El agente debe estar provisto de un método que le permita obtener el valor de Q utilizando los valores más inmediatos ya que, a priori, el agente desconoce los valores de Q de todos los posibles pares. Este método declara que, si una acción lleva al agente a un resultado no deseado, esta acción no debe ser ejecutada, mientras que, si una acción lleva al agente a un resultado favorable, esta acción debe ser ejecutada. Esto se puede representar con refuerzos negativos y positivos respectivamente aplicando distintas magnitudes en ambos casos en función de los bueno o malo que sea el resultado. Además, si todas las acciones de un estado nos proporcionan un resultado negativo, el agente tiene que aprender a no ejecutar acciones que lleven a ese estado, mientras que, si en un estado todas las acciones devuelven un resultado favorecedor, el agente debe aprender a pasar por ese estado si le es posible.

La fórmula más sencilla se compone de:

$$Q(s_t, a_t) = r\,(s_t, a_t) + \gamma \cdot \max\big(Q(s_{t+1}, a_{t+1})\big)$$

El valor máximo de Q se obtiene de la *tabla Q* que almacena tantos valores como combinaciones de estados-acciones haya. Hay tantas columnas como estados y tantas filas como acciones disponibles. Por tanto, dentro de cada estado se puede ejecutar más de una acción, y ese máximo indica el mayor valor dentro de los valores de Q para las acciones dentro de un mismo estado.

Sin embargo, existen 2 parámetros que influyen en el proceso de aprendizaje y en el cálculo del nuevo valor de Q para las tuplas: Alpha y Épsilon.

Alpha, α, representa la tasa de aprendizaje y está comprendida entre 0 y 1. Este parámetro determina si la nueva información obtenida sobre un estado Q tiene más peso sobre la información más antigua ya existente. Este parámetro marca la 'agresividad' con la que se actualizan los valores de Q en la tabla. Cuando su valor está cercano a 1, básicamente, se reemplaza el valor antiguo por el nuevo; mientras que, si está en un valor cercano a 0, se modifica muy levemente el valor antiguo de Q. Por ello, es recomendable empezar el entrenamiento con un valor cercano a 1 ya que inicialmente la tabla Q no tiene valores (todo es cero) y a medida que avanza el entrenamiento, se reduce Alpha hasta alcanzar un valor cercano a 0. Finalmente, modificaría la fórmula resultando en: [14]

$$Q_{new}(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot \Big(\gamma \cdot \max\big(Q(s_{t+1}, a_{t+1})\big)\Big)$$

El último parámetro que tener en cuenta es **Épsilon**, ε, sigue la estrategia de *Epsilon-greedy* dedicada al enfoque de exploración y explotación. Este parámetro va a recibir un valor entre 0 y 1. Cuando el agente elige una acción, se genera un número aleatorio; si este número es menor que Épsilon, se elige una acción aleatoria; y si no, se elige la mejor acción en función de la tabla Q. Inicialmente se empieza potenciando la exploración, es decir, con un valor grande de Épsilon. Pero a medida que el agente progresa por el entrenamiento y aprende qué acciones proporcionan las mejores recompensas a largo

plazo, es lógico reducir el valor de este parámetro a un valor cercano a 0 para poder establecer la estrategia de explotación de la información que ya posee el agente [15].

Se podría considerar un porcentaje de "fallo" en la acción escogida, ejecutando una acción diferente. El objetivo de este parámetro es potenciar la exploración del entrenamiento ya que explora acciones que el agente sabe que tiene poco refuerzo inmediato pero que quizá tenga un refuerzo futuro mayor. Generalmente, este valor empieza en 1 aunque también se puede empezar en 0 según su evolución. Al empezar en 1, o valor cercano a 1, las acciones elegidas por el agente son menos probables a ser elegidas, pero a medida que avanza el entrenamiento, este valor disminuye influyendo cada vez menos hasta que no tiene repercusión en la acción elegida. Aun así, Épsilon no influye en la construcción de la fórmula para obtener el valor de Q, solo en la acción a tomar.

Uno de los problemas más comunes en el Aprendizaje por Refuerzo reside en el riesgo de sobre-aprendizaje o sobreajuste (en inglés *overfitting*) que se puede originar al no cumplirse la segunda condición del conjunto de prueba: el conjunto de entrenamiento no tiene valores similares a las instancias futuras. El problema de *Overfitting* ocurre generalmente cuando la máquina se ajusta únicamente a aprender casos particulares, pero sin la capacidad de reconocer datos nuevos de entrada, es decir, la capacidad de generalizar. Por otra parte, tenemos el *Underfitting* o sub ajuste que se da cuando no hay suficientes instancias en el conjunto de entrenamiento. Para prevenir el *Overfitting* se pueden aplicar prácticas como: utilizar datos variados y equilibrados, hacer uso del conjunto de validación que detecta fácilmente el sobreajuste, un ajuste de parámetros… [16] [17].

2.2.3. EXPLORACIÓN VS EXPLOTACIÓN

Uno de los grandes dilemas a la hora de entrenar a un sistema es la disyuntiva de elegir entre recorrer rutas distintas en busca de mejores resultados o potenciar la situación conocida de la que ya se sabe que va a proporcionar buenos resultados. Estas estrategias se denominan, respectivamente, exploración y explotación.

La exploración consiste en extenderse por el mayor número del espacio de búsqueda para encontrar mejores y más prometedoras soluciones que aún no han sido potenciadas o refinadas. Esta estrategia implica diversificar en gran medida la búsqueda para evitar quedarse estancado en un óptimo local que proporciona un resultado favorable, pero no asegura que sea el óptimo global que otorgue el mejor resultado entre todas las opciones.

La explotación se reduce a un espacio de búsqueda menor con el propósito de mejorar la solución favorable que ya se tiene. En lugar de invertir en la diversificación de búsqueda, se centra en refinar la búsqueda.

Por tanto, podemos referirnos a exploración como una búsqueda global, mientras que la explotación es referida como búsqueda local. Antes del entrenamiento, se debe efectuar un estudio para poder extraer qué estrategia va a beneficiar más al proyecto: potenciar la exploración, potenciar la explotación o establecer un equilibrio entre ambas. Aunque es necesario incluir algo de ambas ya que sin exploración no se podrá encontrar óptimos y sin la explotación no se podrán refinar los resultados obtenidos con el fin de optimizarlos.

Aunque estas estrategias planteen dos enfoques fundamentalmente diferentes, el equilibrio de ambas estrategias produce un efecto beneficioso en muchos dominios dentro del aprendizaje automático. Por ello, en cualquier proyecto se deberían tener patrones balanceados entre las estrategias de exploración y explotación y un control continuo de la gestión de ambas. [18]

Uno de los métodos más comunes que se utiliza en los entrenamientos del sistema para llevar un control preciso de exploración y explotación es la estrategia de *Epsilon-greedy*. Es un algoritmo que añade aleatoriedad cuando se decide entre determinadas opciones dentro del entrenamiento de un modelo. En lugar de elegir siempre la mejor opción disponible, este algoritmo fuerza al modelo a explorar otras opciones en función de una probabilidad ε o elegir la opción que se sabe que es la mejor con una probabilidad de (1-ε). Si se aumenta el valor de ε, la aleatoriedad aumenta a la hora de decidir y, por tanto, se explorará un mayor espacio de búsqueda [15].

2.2.4. APRENDIZAJE EN JUEGOS

El entrenamiento del sistema que juega contra el jugador es uno de los pilares más relevantes a la hora de desarrollar un juego basado en un enfrentamiento máquina-jugador. El resultado obtenido define el modelo, comportamiento, jugabilidad y, sobre todo, la dificultad del juego, modificando así aspectos primordiales en dicha entrega. Por ello, en muchos casos se han implementado distintos valores de dificultad de la IA contra la que se puede jugar para dar al jugador más versatilidad y permitir que se adecúe a un nivel de dificultad apropiado para él/ella. Dependiendo de esta dificultad, la preparación de la IA será diferente haciéndola más inteligente cuanto más elevado sea el nivel de dificultad. Un caso sencillo podría ser un 'juego de lucha' donde si el jugador elige una dificultad menor es más probable que encadene más golpes seguidos mientras que a dificultades mayores la IA será más complicada de golpear y podrá hacer combos al jugador.

Aunque los sistemas más cercanos al público se centran en juegos contra el jugador, existen varios enfoques a la hora de desarrollar una IA que dependen directamente de qué propósito tengan fijado. Fundamentalmente, se tienen dos caminos a seguir a la hora de desarrollar la IA y esto va a ir estrictamente limitado por el objetivo que se quiera alcanzar.

El primer camino se ocupa de crear un sistema capaz de enfrentarse a un jugador y que esté centrado esencialmente en el entretenimiento del jugador o jugadores. La IA debe ser capaz de simular el comportamiento humano para que el enfrentamiento entre la máquina y el jugador sea lo más real posible y, sobre todo, equilibrado para ambos bandos. No se puede elaborar una IA demasiado inteligente que sea muy difícil vencer porque el jugador acabará aborreciendo el juego, pero tampoco se puede crear una IA excesivamente fácil de derrotar ya que tendrá el mismo efecto. Para evitar ese efecto, se debe buscar un equilibrio donde el jugador sea capaz de vencer a la máquina pero que además le suponga un desafío. Sin embargo, cada jugador tiene distinta habilidad para manejarse con los juegos por lo que se añaden niveles de dificultad, previamente mencionados, para que ese equilibrio alcance al máximo número de jugadores. También existen casos en los que

estos niveles de dificultad van aumentando de forma automática y progresiva, donde al principio del juego se tiene un nivel asequible pero que según se avanza, alcanza una complejidad muy elevada. Estos juegos están más centrados en entornos sin ningún tipo de final fijado, donde la meta es conseguir el máximo número de puntos antes de perder; son conocidos como *Bullet Hell* [19]. Como el ejemplo de *Battle Garegga* [20] cuya dificultad aumenta de forma considerable a medida que avanzas en el juego.

El segundo camino está focalizado en la creación de un sistema que gane el mayor número de partidas sin limitarse a objetivos como el entretenimiento de un jugador. Este enfoque existe en competiciones de *bots* o en enfrentamientos entre diferentes sistemas que realizan las mismas tareas para comprobar quién lo hace más rápido y/o más eficiente. Un ejemplo muy conocido es el torneo de IA de *Starcraft SSCAIT (Student Starcraft AI Tournament)* [21] [22] donde estudiantes desarrollan una IA para el juego *Starcraft* para competir con las que han desarrollado otros estudiantes; el objetivo primordial de esta IA es ganar de la forma más eficiente, más rápida y con mejores resultados.

Se podría pensar que las dos alternativas van separadas en función del contrincante en el enfrentamiento, pero hay casos en los que se mezclan y rompen esta separación. Se trata de los sistemas construidos para ser capaces de vencer a cualquier humano en una tarea específica, en este caso, juegos. Uno de los casos más famosos es *DeepBlue* [23], una computadora creada por IBM que consiguió vencer al campeón mundial de ajedrez *Garry Kasparov* en su segundo enfrentamiento en 1997. Otro ejemplo similar al anterior es *AplhaGo*, que consiguió ganar a diferentes jugadores profesionales del juego *Go* como los enfrentamientos contra *Fan Hui* en 2015 y *Lee Sedol* en 2016 utilizando redes de neuronas y árboles de búsqueda Montecarlo entre otras herramientas [24] [25]. Además, Google, después de su éxito con la creación de *AlphaGo*, consiguió superar las capacidades de la máquina de IBM con el desarrollo de *AlphaZero*. Esta IA consiguió vencer en 2016 a *Stockfish*, un importante motor de ajedrez de código abierto que se puso en cabeza en la temporada 11 de la TCEC (*Top Chess Engine Championship*) [26] [27] [28]. Utilizando un método de auto-aprendizaje, en menos de 4 horas consiguió establecer la IA que fue capaz de lograr, en 100 partidas intercalando blancas y negras, la victoria ante *Stockfish* en 28 ocasiones y hacer tablas en las 72 partidas restantes [29] [30].

2.3. ARTÍCULOS Y PROYECTOS SIMILARES

Esta sección está dedicada al estudio de proyectos ajenos que guardan cierta relación con este proyecto ya sea por el ámbito de la Inteligencia Artificial, por los videojuegos o por ambas partes. Aun así, se han elegido proyectos donde la IA de los videojuegos sea el centro de estudio.

Con el objetivo de efectuar una comparativa ente los proyectos en sí y este proyecto, primero se expone un resumen de cada uno de los elegidos y posteriormente se contrastan mediante determinados aspectos presentados de forma tabular.

El primer proyecto para analizar: "*Learning to play Tetris applying reinforcement learning methods*" desarrollado por *Friedhelm Schwenker* (2008) está centrado en desarrollar una IA que sea capaz de jugar de forma automática un partida de Tetris, un

juego de puzzles donde diferentes piezas caen de la parte superior de la pantalla y el jugador puede mover las piezas o rotarlas para dejarlas en el lugar y la posición que desea para eliminar líneas que ya están completas [31]. Para ello se empleó Aprendizaje por Refuerzo y Ecuaciones de *Bellman*.

En el siguiente proyecto llamado "*Designing a Reinforcement Learning-based Adaptive AI for Large-Scale Strategy Games*" de *Charles Madeira*, *Vincent Corruble* y *Geber Ramalho* (2006) se estudia la inserción de una IA en juegos de estrategia utilizando Aprendizaje por Refuerzo. El juego empleado es *BattleGround*^{TM}, un juego de guerra de grandes dimensiones con una jugabilidad por turnos entre 2 jugadores representando batallas históricas.

El proyecto a continuación, de *Randle Oluwarotimi Abayomi* y *Lall Manosh* (2013) llamado "*An Overview of Supervised Machine Learning Techniques in Evolving Mancala Game player*" se centra en un agente automático utilizando distintas técnicas de aprendizaje supervisado para que sea capaz de jugar al juego de tablero *Mancala*. Este juego tiene determinados agujeros donde colocar las piezas, recogiéndolas y moviéndolas entre los agujeros correspondientes para conseguir puntos acabando cuando los agujeros no tienen ninguna pieza [32] [33].

Referencia	**Tipo de juego**	**¿Humano vs IA?**	**Aprendizaje**	**Algoritmo(s)**	**Enfoque**
Friedhelm Schwenker (2008)	Puzles	No	Refuerzo	*Bellman Equations*	Mejorar estrategias de políticas
Charles Madeira, *Vincent Corruble* y *Geber Ramalho* (2006)	Estrategia por turnos	Sí	Refuerzo	*Gradient-descent Sarsa*	Comprobar efectividad del algoritmo en juegos de estrategia de gran escala
Randle Oluwarotimi Abayomi y *Lall Manosh* (2013)	Juego de tablero	Sí	Supervisado	-Minimax -Co-evolutions -Linear Discriminant -Genéticos	Comparar técnicas de aprendizaje supervisado
Proyecto propio	Combate por turnos	Sí	Refuerzo	*Q-Learning*	Comprobar efectividad del algoritmo en juegos por turnos

Tabla 1: Comparación de proyecto similares

3. ANÁLISIS

Esta sección cubre todas las tareas relacionadas con el análisis del proyecto en cuestión. Se empieza analizando el alcance del análisis, es decir, cuáles son las tareas principales del proyecto y qué progresión se debe seguir. A continuación, se estudian las alternativas que hay a la hora de realizar cada tarea, en caso de haber varias herramientas o estrategias viables para una misma.

Posteriormente, se plantea primero una lista de requisitos que debe cumplir el sistema y luego los casos de uso existentes completando dichas secciones con las correlaciones entre ambas.

Y como última tarea del apartado de análisis, se establece una batería de pruebas para cubrir todos los posibles errores en el sistema.

3.1. ALCANCE Y TAREAS

Esta subsección del apartado de Análisis se centra en las tareas y funcionalidades del sistema. Para ello se debe desmontar el proyecto en las distintas partes que tiene y las acciones necesarias para alcanzar esas funcionalidades. Principalmente, el proyecto se divide en dos apartados bien diferenciados: la creación del motor de juego y el desarrollo de una IA que actúe en él.

El motor del juego varía su tamaño según las distintas funcionalidades existentes en la jugabilidad de este. Para no entrar en gran detalle en este apartado, se pueden separar estas funcionalidades en: movimiento por un mapa y combate por turnos. Por tanto, las funcionalidades sería desarrollar ambas ramas del motor del juego.

La separación de tareas dentro del desarrollo de la IA se extiende más que el primer apartado. Inicialmente, se debe crear un agente secundario capaz de simular las acciones de un humano para poder entrenar al sistema de forma automática más adelante. Después, teniendo ya el agente humano que pueda actuar de forma automática en un enfrentamiento contra la máquina, se necesita preparar un entorno de entrenamiento que enfrente un número de ciclos a ambos agentes, humano y máquina, en distintas situaciones. Antes de poder realizar la conexión entre el aprendizaje y el motor de juego, se debe realizar otro entorno de enfrentamiento en el que se estudie el porcentaje de victorias de la IA para, finalmente, saber qué modelo implementar en el sistema de combate del motor del juego.

Para resumir estas tareas, obtendríamos una lista:

- Implementar movimiento por un mapa en el motor de juego
- Implementar combate por turnos en el motor de juego
- Capturar acciones de usuarios en entorno simple para crear un agente humano
- Entrenar al sistema con ese agente
- Extraer resultados de los distintos entrenamientos
- Implementar el sistema de decisión de los enemigos en el sistema de combate del motor de juego

3.2. ESTUDIO DE ALTERNATIVAS

Con el objetivo de realizar las tareas de forma eficaz y mejorar los posibles resultados, se añade este apartado que está centrado en realizar un estudio de las distintas alternativas que existen para ejecutar una misma tarea. Para elegir la mejor opción entre todas las alternativas, se efectúa una investigación de cada una de ellas que permite ver cuál se adapta mejor a la idea del proyecto.

El desarrollo de todas las tareas mencionadas en el alcance del proyecto se puede conseguir de forma muy sencilla por la baja cantidad de componentes de trabajo. Al no utilizar herramientas físicas externas y realizar las tareas con conocimientos propios y en muchos casos creativos, solo se analizan las herramientas de desarrollo.

En el progreso de crear un sistema como el de este proyecto, es clave prestar atención a qué herramientas van a encajar más por su versatilidad, estructuración y facilidad para realizar tareas determinadas. Estas tareas elegidas deben centrarse en este proyecto ya que para otros proyectos puede que no sean útiles.

Las herramientas de desarrollo se bifurcan en la creación de un agente humano y el desarrollo del motor del juego.

3.2.1. AGENTE HUMANO

El agente humano va a ser aquel encargado de simular las acciones que efectúe un jugador. Esto se emplea para poder realizar entrenamientos de forma automática y más veloz que si lo hiciese un jugador de forma manual. Pero ese agente se puede obtener con metodologías variadas que no tienen por qué tener relación entre ellas.

Un método sencillo para generar un modelo de agente humano es la creación de **reglas** que analicen el entorno para extraer en qué situación se encuentra el jugador y devuelvan una salida en función de esas condiciones. Para ello, no es necesario que ningún usuario juegue, sino que se crea un programa externo que vaya por todas las posibilidades de los factores que afectan a la decisión y devuelva un valor de salida determinado. Esta metodología abarca el total de posibles situaciones que puede encontrarse el jugador, pero solo es eficiente en aquellos casos donde las combinaciones posibles sean reducidas, ya que, en caso contrario, se emplearía demasiado tiempo en la búsqueda de la situación en la que se encuentra.

Otro método capaz de generar este agente humano es el uso de herramientas de aprendizaje automático que analicen el entorno y que en función de sus valores devuelvan la clase, en este caso la acción a ejecutar. Estas herramientas son conocidas como clasificadores.

La fuente más conocida de estas herramientas es la Universidad de Waikato que han estado años trabajando en programas de uso libre (*open source*) para efectuar tareas de aprendizaje automático entre las que se incluyen clasificación y regresión. Para el caso particular de este proyecto, se debe centrar en clasificación más que en el resto de las actividades. Entre los programas desarrollados por la universidad, se encuentra el popular

WEKA que se trata del primer software libre de Waikato dedicado al aprendizaje automático y minería de datos desarrollado en el lenguaje de Java y distribuido bajo la licencia *GNU-GPL*. Para facilitar el uso de su herramienta, la Universidad de Waikato proporciona a todo usuario que esté interesado en usarla una gran cantidad de documentación, preguntas frecuentes, y cursos. WEKA está provisto de muchos parámetros variados que facilitan al usuario una mejor adaptación a su modelo de clasificación, regresión o agrupación [34] [35].

Otra herramienta capaz de crear un clasificador es **MEKA**, basada en el *toolkit* de WEKA, que proporciona una implementación de uso libre de aprendizaje y evaluación con etiquetas múltiples (*multi-label*) para predecir múltiples variables de salida para cada instancia de entrada. Sin embargo, el uso de este programa puede no ser el más apropiado para el proyecto en cuestión. Cuando trabajamos con un clasificador de etiquetas simples, devuelve una única salida por cada instancia y se le asocia un único valor. Por otro lado, al trabajar con etiquetas múltiples puede que haya múltiples salidas o puede que la salida que haya pueda tener una combinación de los posibles valores. Como el proyecto requiere una salida simple sin combinación de valores, este software no es recomendable usarlo. Si se hubiese utilizado una combinación de acciones de todos los agentes humanos, en lugar de ejecutar una acción por cada agente de forma separada, esta herramienta habría sido de utilidad [36].

Por tanto, las dos primeras herramientas son válidas para la creación del agente humano. Aunque las condiciones favorecen a la estructuración por sistema de reglas, el uso de un clasificador puede también proporcionar resultados igual de favorables si se construye apropiadamente.

3.2.2. DESARROLLO DEL MOTOR DE JUEGO

La decisión para elegir el lenguaje de programación para el motor de juego se cerró en dos opciones: Python o Java. Inicialmente, solo se consideró el uso de Python por su sencilla interfaz y facilidad para determinadas tareas, pero como hay ciertas herramientas de clasificación que están desarrollada en Java también se tuvo en cuenta para la fácil conexión entre ambas partes.

Aun así, los beneficios que proporciona Python sobrepasan a usar Java. El motor extrae la información del estado del juego, jugadores y enemigos a partir de archivos con un formato determinado. Por lo que la lectura de estos archivos sirve para obtener información y la escritura de estos sirve para actualizarla. Python en sus versiones más recientes tiene una gran variedad de librerías que permiten al usuario realizar estas escrituras y lecturas de forma rápida y en pocas líneas de código, sobre todo para archivos cuyas extensiones sean *csv* que puedan ocasionar problemas de formato. Además, la familiaridad con este programa es mayor que con Java.

3.3. REQUISITOS

Teniendo en cuenta el análisis realizado en los apartados anteriores de esta sección, este apartado se centra en la presentación de todos los requisitos del sistema junto con sus atributos correspondientes.

La división de estos requisitos se separa en requisitos funcionales que marquen qué va a realizar el sistema y no funcionales que determinen cómo va a llevar a cabo esas tareas.

Para la exposición de los requisitos se empleará un modelo tabular que constará de los siguientes apartados:

ReqID: Un código alfanumérico único que identifica al requisito. Para los requisitos funcionales se seguirá un formato de RF-XX y para los no funcionales RNF-XX, donde XX representa el código numérico de cada uno empezando por 01.

Título: Descripción corta del requisito.

Descripción: Descripción más extensa del requisito.

Prioridad: Importancia que tiene el requisito en el sistema. Puede obtener valores como Alta, Media o Baja.

ReqID	*RF-00*
Título	
Descripción	
Prioridad	

Tabla 2: Ejemplo de requisito

3.3.1. REQUISITOS FUNCIONALES

ReqID	*RF-01*
Título	Creación de número de personajes
Descripción	Genera el número de personajes en función de lo marcado por el/los jugadores
Prioridad	Alta

*Tabla 3: Requisito **RF-01**: Creación de número de personajes*

ReqID	*RF-02*
Título	Creación de nombre de personaje
Descripción	Genera el nombre asociado al personaje que el jugador elija para hacer referencia en futuras situaciones
Prioridad	Baja

*Tabla 4: Requisito **RF-02**: Creación de nombre de personaje*

ReqID	*RF-03*
Título	Creación de clase de personaje
Descripción	Genera la clase del personaje que el jugador elija para regir la jugabilidad de ese personaje
Prioridad	Alta

*Tabla 5: Requisito **RF-03**: Creación de clase de personaje*

ReqID	*RF-04*
Título	Uso de atributos iniciales de personaje
Descripción	Aumenta los atributos que el jugador elija hasta 4 veces de 1 en 1
Prioridad	Media

*Tabla 6: Requisito **RF-04**: Uso de atributos iniciales de personaje*

ReqID	*RF-05*
Título	Elección de arma inicial de personaje
Descripción	Incorpora el arma elegida por el jugador al equipamiento de su personaje
Prioridad	Media

*Tabla 7: Requisito **RF-05**: Elección de arma inicial de personaje*

ReqID	*RF-06*
Título	Elección de misión
Descripción	Se establece la partida en función de la opción de misión elegida por los jugadores en conjunto
Prioridad	Alta

*Tabla 8:Requisito **RF-06**: Elección de misión*

ReqID	*RF-07*
Título	Cambiar de zona
Descripción	Se desplaza a todos los jugadores a la zona que hayan elegido en conjunto para moverse dentro del mapa elegido en la misión
Prioridad	Alta

*Tabla 9: Requisito **RF-07**: Cambiar de zona*

ReqID	*RF-08*
Título	Generación de enemigos
Descripción	Se generan los tipos de enemigos de forma aleatoria al entrar en una zona nueva (tantos enemigos como jugadores). Se generan tantos como jugadores haya.
Prioridad	Alta

*Tabla 10: Requisito **RF-08**: Generación de enemigos*

ReqID	*RF-09*
Título	Generación de cofres
Descripción	Al acabar un enfrentamiento en una zona, se generan los cofres en esa zona de forma aleatoria entre 0 y el número de personajes
Prioridad	Baja

*Tabla 11: Requisito **RF-09**: Generación de cofres*

ReqID	*RF-10*
Título	Apertura de cofre
Descripción	Un jugador abre un cofre de la zona para revelar su contenido, reduciendo el número total de cofres sin abrir de la zona
Prioridad	Baja

*Tabla 12: Requisito **RF-10**: Apertura de cofre*

ReqID	*RF-11*
Título	Generación de arma de cofre
Descripción	La apertura de un cofre genera un arma aleatoria de la lista de arsenal para dar la opción de poder equipársela a un personaje
Prioridad	Baja

*Tabla 13: Requisito **RF-11**: Generación de arma de cofre*

ReqID	*RF-12*
Título	Generación de pociones de cofre
Descripción	La apertura de un cofre genera un número de pociones aleatorio entre 0 y el número de jugadores
Prioridad	Baja

*Tabla 14: Requisito **RF-12**: Generación de pociones de cofre*

ReqID	*RF-13*
Título	Equipación de arma de cofre
Descripción	Un jugador se equipa el arma que haya aparecido en el cofre siendo esa su nueva arma y soltando la suya actual en el suelo de la zona
Prioridad	Media

*Tabla 15: Requisito **RF-13**: Equipación de arma de cofre*

ReqID	*RF-14*
Título	Equipación de arma del suelo
Descripción	Un jugador se equipa un arma que esté en el suelo de la zona siendo esa su nueva arma y soltando la suya actual en el suelo
Prioridad	Media

*Tabla 16: Requisito **RF-14**: Equipación de arma del suelo*

ReqID	*RF-15*
Título	Recogida de pociones
Descripción	Un jugador con menos de 2 pociones en su inventario recoge una poción aumentando en 1 la cantidad de pociones propias y reduciendo en 1 las pociones del cofre
Prioridad	Media

*Tabla 17: Requisito **RF-15**: Recogida de pociones*

ReqID	*RF-16*
Título	Ataque a enemigo
Descripción	El jugador ataca al enemigo aplicando una cantidad de puntos de daño y reduciendo la vida de ese enemigo el mismo número de puntos
Prioridad	Alta

*Tabla 18: Requisito **RF-16**: Ataque a enemigo*

ReqID	*RF-17*
Título	Entrega de experiencia por ataque
Descripción	Se otorga una cantidad de experiencia igual al daño que haga un jugador a un enemigo
Prioridad	Media

*Tabla 19: Requisito **RF-17**: Entrega de experiencia por ataque*

ReqID	*RF-18*
Título	Entrega de experiencia por derrota de enemigo
Descripción	Se otorga una cantidad fija de 50 puntos de experiencia a todos los jugadores vivos cuando un enemigo es derrotado
Prioridad	Media

*Tabla 20: Requisito **RF-18**: Entrega de experiencia por derrota de enemigo*

ReqID	*RF-19*
Título	Subida de nivel
Descripción	Cuando la experiencia llega a un valor determinado o superior, se resetea al valor sobrante aumentando en 1 el nivel del jugador
Prioridad	Media

*Tabla 21: Requisito **RF-19**: Subida de nivel*

ReqID	*RF-20*
Título	Uso de atributo por nuevo nivel
Descripción	Cuando un personaje sube de nivel se aumenta en 1 el valor del atributo que el jugador elija.
Prioridad	Media

*Tabla 22: Requisito **RF-20**: Uso de atributo por nuevo nivel*

ReqID	*RF-21*
Título	Derrota de enemigo
Descripción	El enemigo atacado cuyos puntos de vida estén en un valor de 0 o inferior es eliminado y no puede ejecutar más acciones
Prioridad	Alta

*Tabla 23: Requisito **RF-21**: Derrota de enemigo*

ReqID	*RF-22*
Título	Derrota de jugador
Descripción	El personaje atacado cuyos puntos de vida estén en un valor de 0 o inferior es eliminado y no puede ejecutar más acciones
Prioridad	Alta

*Tabla 24: Requisito **RF-22**: Derrota de jugador*

ReqID	*RF-23*
Título	Uso de poción
Descripción	El personaje cuya vida no esté al máximo gasta una poción reduciendo su número de pociones en 1 y aumentando su vida hasta 5 puntos sin superar el valor de vida máxima
Prioridad	Alta

*Tabla 25: Requisito **RF-23**: Uso de poción*

ReqID	*RF-24*
Título	Ataque a personaje
Descripción	El enemigo elige al jugador para atacar causando una cantidad de puntos de daño y reduciendo la vida de ese personaje el mismo número de puntos
Prioridad	Alta

*Tabla 26:Requisito **RF-24**: Ataque a personaje*

3.3.2. REQUISITOS NO FUNCIONALES

ReqID	*RNF-01*
Título	Sistema operativo
Descripción	El sistema funciona en Windows, Linux y iOS
Prioridad	Alta

*Tabla 27:Requisito **RNF-01**: Sistema operativo*

ReqID	*RNF-02*
Título	Lenguaje de programación
Descripción	El sistema está desarrollado en Python
Prioridad	Alta

*Tabla 28: Requisito **RNF-02**: Lenguaje de programación*

ReqID	*RNF-03*
Título	Librerías de Python
Descripción	Para efectuar ciertas funcionalidades se importan librerías como: *random, csv y math*
Prioridad	Media

*Tabla 29: Requisito **RNF-03**: Librerías de Python*

3.4. CASOS DE USO

Los casos de uso recogen todas las actividades que puede llevar a cabo un actor, en este caso los jugadores, dentro del sistema y sus interacciones con él. Se utilizan para controlar las acciones que ocurren durante el proceso y las funcionalidades del sistema.

Más adelante, se utilizarán para poder conectar las funcionalidades del sistema con las posibles actividades que pueda realizar un actor.

Los casos de uso estarán estructurados de forma tabular con los siguientes atributos:

- **CasoID**: Un código alfanumérico único que identifica al caso de uso. Se estructura como CU-XX siendo XX un valor numérico que empieza en 01
- **Título**: Breve descripción del caso de uso
- **Actor**: Entidad externa del sistema que interactúa con él en el caso de uso y demanda una funcionalidad. En este caso, habrá dos posibles valores: un jugador en particular o, en caso de haber más de uno, los jugadores que tendrán que discutir qué acción efectuar.
- **Pre-condición:** Situación del sistema previa a realizar el caso de uso
- **Post-condición:** Consecuencia o consecuencias del caso de uso
- **Flujo normal:** Desarrollo del caso de uso

CasoID	*CU-00*
Título	
Actor	
Pre-condición	
Post-condición	
Flujo normal	

Tabla 30: Caso de Uso de ejemplo

CasoID	*CU-01*
Título	Elegir número de jugadores
Actor	Jugadores (acción conjunta)
Pre-condición	Los jugadores desean empezar una partida nueva
Post-condición	Los bucles y estructuras de personajes están definidas
Flujo normal	Los jugadores ejecutan el programa y marcan el número de jugadores que van a jugar hasta un máximo de 3. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 31: Caso de Uso **CU-01**: Elegir número de jugadores*

CasoID	*CU-02*
Título	Elegir nombre de personaje
Actor	Jugador
Pre-condición	Se ha establecido el número de jugadores
Post-condición	El sistema tiene el nombre del jugador establecido para referirse a él/ella en próximas alusiones
Flujo normal	El sistema pide que el jugador introduzca el nombre deseado El jugador escribe el nombre. Si no está repetido se procede al siguiente paso, en caso de estarlo debe escribir otro.

*Tabla 32: Caso de Uso **CU-02**: Elegir nombre de personaje*

CasoID	*CU-03*
Título	Elegir clase de personaje
Actor	Jugador
Pre-condición	Ya se ha marcado el nombre del personaje
Post-condición	El sistema tiene la clase del jugador establecida para tener en cuenta el daño causado y recibido
Flujo normal	El sistema pide que el jugador introduzca la clase deseada El jugador introduce un valor entre 1 y 3. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 33: Caso de Uso **CU-03**: Elegir clase de personaje*

CasoID	*CU-04*
Título	Gastar puntos de atributos iniciales
Actor	Jugador
Pre-condición	La clase está definida y, por tanto, los valores de los atributos de inicio
Post-condición	Los atributos del personaje aumentan en función de los elegidos por el jugador
Flujo normal	Durante cuatro iteraciones el sistema plantea al jugador los 3 posibles atributos que puede mejorar. Para cada iteración, el jugador introduce un valor entre 1 y 3. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 34: Caso de Uso **CU-04**: Gastar puntos de atributos iniciales*

CasoID	*CU-05*
Título	Elegir arma inicial
Actor	Jugador
Pre-condición	Se han aumentado los atributos iniciales
Post-condición	El personaje tiene un arma equipada
Flujo normal	El sistema presenta al jugador las 3 posibles armas a elegir generadas de forma aleatoria de la lista del arsenal. El jugador introduce un valor entre 1 y 3. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 35: Caso de Uso **CU-05**: Elegir arma inicial*

CasoID	*CU-06*
Título	Elegir misión
Actor	Jugadores (acción conjunta)
Pre-condición	Todos los personajes están creados, preparados y equipados
Post-condición	Empieza la partida en la zona inicial dependiendo de la misión elegida.
Flujo normal	El sistema presenta a los jugadores las 2 posibles misiones. Se introduce un valor de 1 ó 2. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 36: Caso de Uso **CU-06:** Elegir misión*

CasoID	*CU-07*
Título	Moverse de zona
Actor	Jugadores (acción conjunta)
Pre-condición	Todos los enemigos de la zona en la que se encuentran han sido derrotados
Post-condición	La zona en la que se encuentran los jugadores cambia a la elegida
Flujo normal	El sistema presenta a los jugadores entre 1 y 3 posibles destinos. Se introduce un valor entre esos valores. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 37: Caso de Uso **CU-07**: Moverse de zona*

CasoID	*CU-08*
Título	Atacar a enemigo
Actor	Jugador
Pre-condición	El personaje y enemigo al que atacar están vivos. Es el turno del jugador
Post-condición	El enemigo tiene tanta vida como la diferencia de sus puntos de vida anteriores y el daño efectuado por el jugador. Si es de 0 o menos, el enemigo es derrotado.
Flujo normal	El sistema presenta al jugador los enemigos vivos que hay. El jugador introduce el número del enemigo que quiere atacar. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 38: Caso de Uso **CU-08**: Atacar a enemigo*

CasoID	*CU-09*
Título	Tomar poción
Actor	Jugador
Pre-condición	Es el turno del jugador. Tiene menos puntos de vida que sus puntos máximos. Tiene al menos una poción
Post-condición	El personaje recupera hasta 5 puntos de vida sin superar la barrera de vida máxima
Flujo normal	En su turno, el jugador elige la opción de tomarse la poción en lugar de la acción de atacar.

*Tabla 39: Caso de Uso **CU-09:** Tomar poción*

CasoID	*CU-10*
Título	Abrir cofre
Actor	Jugadores (acción conjunta)
Pre-condición	Todos los enemigos de la zona en la que se encuentran han sido derrotados. Hay al menos un cofre sin abrir
Post-condición	Se muestran el contenido del cofre. Se reduce el número de cofres sin abrir en la zona 1 unidad
Flujo normal	Después del enfrentamiento, se elige la opción de abrir un cofre y muestra los objetos que contenía.

*Tabla 40: Caso de Uso **CU-10**: Abrir cofre*

CasoID	*CU-11*
Título	Equipar arma del cofre
Actor	Jugadores (acción conjunta)
Pre-condición	Se ha abierto un cofre.
Post-condición	El personaje obtiene esa arma como propia. El arma que llevaba equipada se queda en el suelo de esa zona
Flujo normal	Después de abrir el cofre, se muestra el arma que sale de él y se pregunta a los jugadores si quieren el arma. En caso afirmativo, se les pide que elijan quién se queda el arma. Se introduce un valor entre 1 y el número de jugadores vivos. Si es un input válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 41: Caso de Uso **CU-11**: Equipar arma de cofre*

CasoID	*CU-12*
Título	Soltar arma del cofre
Actor	Jugadores (acción conjunta)
Pre-condición	Se ha abierto un cofre.
Post-condición	El arma del cofre se queda en el suelo de esa zona
Flujo normal	Después de abrir el cofre, se muestra el arma que sale de él y se pregunta a los jugadores si quieren el arma. En caso negativo, nadie obtiene el arma y ésta se queda en el suelo.

*Tabla 42: Caso de Uso **CU-12**: Soltar arma del cofre*

CasoID	*CU-13*
Título	Equipar arma del suelo
Actor	Jugadores (acción conjunta)
Pre-condición	Todos los enemigos de la zona en la que se encuentran han sido derrotados. Hay al menos un arma en el suelo de la zona.
Post-condición	El personaje obtiene esa arma como propia. El arma que llevaba equipada se queda en el suelo de esa zona
Flujo normal	Después del enfrentamiento, se elige la opción de equipar arma del suelo y se pide a los jugadores que elijan quién se la queda. Se introduce un valor entre 1 y el número de jugadores vivos. Si es un input válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 43: Caso de Uso **CU-13:** Equipar arma del suelo*

CasoID	*CU-14*
Título	Coger poción
Actor	Jugadores (acción conjunta)
Pre-condición	Se ha abierto un cofre y se ha asignado el arma que contenía a un personaje o al suelo. Hay al menos una poción proveniente de ese cofre para recoger. El personaje que la coge tiene menos de 2 pociones en el inventario.
Post-condición	Aumenta en uno el número de pociones que tiene el personaje elegido en el inventario.
Flujo normal	Después de abrir un cofre y de asignar el arma que sale a un jugador o dejarla en el suelo, se elige quién va a recoger la poción. Se introduce un valor entre 1 y el número de jugadores vivos. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 44: Caso de Uso **CU-14**: Coger poción*

CasoID	*CU-15*
Título	Aumentar punto de atributo
Actor	Jugador
Pre-condición	El jugador ha subido de nivel.

Post-condición	Aumenta en uno el valor del atributo elegido por el jugador
Flujo normal	Después de subir de nivel, el sistema presenta al jugador los 3 atributos disponibles para aumentar. El jugador introduce un valor entre 1 y 3. Si es un *input* válido se procede al siguiente paso, en caso de no serlo, el sistema pide que se introduzca uno válido.

*Tabla 45: Caso de Uso **CU-15**: Aumentar punto de atributo*

Además de la exposición de los posibles casos de uso que existen entre los diferentes actores y el sistema, se debe establecer una conexión entre éstos y los requisitos funcionales del sistema para saber qué tareas están incluidas en cada una de las posibles acciones.

Para ello, se recurre a la matriz de trazabilidad que se constituye por filas de requisitos funcionales y columnas de casos de uso. Para los requisitos que estén involucrados en un caso de uso, se marca esa celda coincidente con una X. Al tener más requisitos, muchos casos de uso involucran más de un requisito implícito.

	CU-01	CU-02	CU-03	CU-04	CU-05	CU-06	CU-07	CU-08	CU-09	CU-10	CU-11	CU-12	CU-13	CU-14	CU-15
RF-01	X														
RF-02		X													
RF-03			X												
RF-04				X											
RF-05					X										
RF-06						X									
RF-07							X								
RF-08							X								
RF-09							X								
RF-10										X					
RF-11										X					
RF-12										X					
RF-13											X	X			
RF-14												X	X		
RF-15														X	
RF-16								X							
RF-17								X							
RF-18								X							
RF-19								X							
RF-20															X
RF-21								X							
RF-22															
RF-23									X						
RF-24															

Tabla 46: Matriz de Trazabilidad: Requisitos y Casos de Uso

3.5. BATERIA DE PRUEBAS

Con el objetivo de asegurar los requisitos del sistema y de comprobar su apropiado funcionamiento, se establece una batería de pruebas que consta de varias partes que cubren la totalidad de las partes en las que puede haber algún error. Este apartado sólo se centra en el planteamiento de las pruebas ya que la realización de estas pruebas se hará en la sección de Implementación y Evaluación.

Solo se analizan las pruebas dinámicas, es decir, aquellas que requieren la ejecución del programa para efectuarlas.

Para estructurar las distintas pruebas realizadas se utiliza un modelo tabular que consta de los siguientes elementos:

- **PruebaID**: Un código alfanumérico único que identifica a la prueba. Se estructura como PR-XX siendo XX un valor numérico que empieza en 01
- **Nombre:** Breve alusión al enfoque de la prueba
- **Procedimiento:** Pasos a seguir para ejecutar la prueba
- **Resultado esperado:** Lo que se espera que dé como resultado la prueba
- **Satisfactoria:** 'Sí' en caso de serlo, 'No' en caso contrario. No se marca en esta sección del documento todavía.
- **Requisitos relacionados:** Contiene, en función del formato definido en la sección de requisitos, el ID del requisito que es probado.

PruebaID	*PR-00*
Nombre	
Procedimiento	
Resultado esperado	
Satisfactoria	
Requisitos relacionados	

*Tabla 47: Prueba **PR-00:** Ejemplo*

PruebaID	*PR-01*
Nombre	*Input* fuera de límite numérico establecido
Procedimiento	Cuando al jugador o jugadores se les pide elegir una opción numérica entre las presentadas, se elige un valor fuera de ese rango
Resultado esperado	Mensaje de error: "No es válido, introduzca otro". Se vuelve a esperar *input* del jugador o jugadores
Satisfactoria	
Requisitos relacionados	*RF-01, RF-03, RF-04, RF-05, RF-06, RF-07, RF-10, RF-13, RF-14, RF-15, RF-16, RF-20, RF-23*

*Tabla 48: Prueba **PR-01**: Input fuera de límite numérico establecido*

PruebaID	*PR-02*
Nombre	*Input* alfanumérico cuando se espera un número
Procedimiento	Cuando al jugador o jugadores se les pide elegir una opción numérica entre las presentadas, se recibe un valor alfanumérico
Resultado esperado	Mensaje de error: "Eso no es un número, introduzca uno válido". Se vuelve a esperar *input* del jugador o jugadores
Satisfactoria	
Requisitos relacionados	*RF-01, RF-03, RF-04, RF-05, RF-06, RF-07, RF-10, RF-13, RF-14, RF-15, RF-16, RF-20, RF-23*

*Tabla 49: Prueba **PR-02**: Input alfanumérico cuando se espera un número*

PruebaID	*PR-03*
Nombre	Utilizar nombre de personaje ya existente
Procedimiento	Cuando al jugador se le pide dar nombre a su personaje, se utiliza uno ya registrado
Resultado esperado	Mensaje de error: "¡Ese nombre ya está cogido! Elige otro". Se vuelve a esperar *input* del jugador
Satisfactoria	
Requisitos relacionados	*RF-02*

*Tabla 50: Prueba **PR-03:** Utilizar nombre de personaje ya existente*

PruebaID	*PR-04*
Nombre	Abrir cofre cuando no hay ninguno disponible
Procedimiento	Se elige la opción de abrir cofre cuando se muestra "(0 cofres disponibles)" en dicha opción
Resultado esperado	Mensaje de error: "No es válido, introduzca otro". Se vuelve a esperar *input* del jugador
Satisfactoria	
Requisitos relacionados	*RF-10*

*Tabla 51: Prueba **PR-04**: Abrir cofre cuando no hay ninguno disponible*

PruebaID	*PR-05*
Nombre	Recoger poción cuando se tienen 2
Procedimiento	Se elige la opción de recoger poción, pero solo aparecen aquellos jugadores que tienen menos de 2 pociones en el inventario.
Resultado esperado	El nombre del personaje que tiene 2 pociones no aparece en el listado
Satisfactoria	
Requisitos relacionados	*RF-15*

*Tabla 52: Prueba **PR-05**: Recoger poción cuando se tienen 2*

PruebaID	*PR-06*
Nombre	Generar enemigos en nueva zona
Procedimiento	Cuando se cambia de zona, se generan enemigos de distintos tipos
Resultado esperado	El número de enemigos generados coincide con el número de jugadores
Satisfactoria	
Requisitos relacionados	*RF-08*

*Tabla 53: Prueba **PR-06**: Generar enemigos en nueva zona*

PruebaID	*PR-07*
Nombre	Generar cofres en nueva zona
Procedimiento	Cuando se cambia de zona, se generan un número aleatorio de cofres, entre 0 y el número de jugadores
Resultado esperado	El número de cofres generados está dentro de ese rango
Satisfactoria	
Requisitos relacionados	*RF-10*

*Tabla 54: Prueba **PR-07**: Generar cofres en nueva zona*

PruebaID	*PR-08*
Nombre	Generar pociones de cofre
Procedimiento	Cuando se abre un cofre, se generan un número aleatorio de pociones, entre 0 y el número de jugadores
Resultado esperado	El número de pociones generadas está dentro de ese rango
Satisfactoria	
Requisitos relacionados	*RF-12*

*Tabla 55: Prueba **PR-08:** Generar pociones de cofre*

PruebaID	*PR-09*
Nombre	Generar arma de cofre
Procedimiento	Cuando se abre un cofre, se genera un arma de la lista de arsenal
Resultado esperado	Se muestra el arma que pertenece a la lista de arsenal
Satisfactoria	
Requisitos relacionados	*RF-11*

*Tabla 56: Prueba **PR-09**: Generar arma de cofre*

PruebaID	*PR-10*
Nombre	Cambiar de arma
Procedimiento	Cuando se coge un arma nueva, se cambia el arma que se tenía por la nueva
Resultado esperado	El arma recogida se establece como arma propia y el arma que se tenía antes se deja en el suelo de la zona
Satisfactoria	
Requisitos relacionados	*RF-13, RF-14*

*Tabla 57: Prueba **PR-10**: Cambiar de arma*

PruebaID	*PR-11*
Nombre	Tomarse poción con toda la vida
Procedimiento	Marcar opción de tomarse poción cuando los puntos de vida están al máximo
Resultado esperado	Mensaje de error: "Ya tienes toda la vida, introduzca otro valor". Se vuelve a esperar *input* del jugador
Satisfactoria	
Requisitos relacionados	*RF-23*

*Tabla 58: Prueba **PR-11**: Tomarse poción con toda la vida*

PruebaID	*PR-12*
Nombre	Enemigo derrota a un personaje
Procedimiento	Un enemigo ataca a un personaje y lo mata
Resultado esperado	El personaje muere y no puede ejecutar más acciones.
Satisfactoria	
Requisitos relacionados	*RF-16, RF-22*

*Tabla 59: Prueba **PR-12**: Atacar cuando el personaje está derrotado*

PruebaID	*PR-13*
Nombre	Atacar a enemigo derrotado
Procedimiento	Elegir la opción de atacar a ese enemigo
Resultado esperado	El enemigo que esté derrotado no puede efectuar ninguna acción ni ser objetivo de ningún ataque
Satisfactoria	
Requisitos relacionados	*RF-16, RF-21*

*Tabla 60: Prueba **PR-13**: Atacar a enemigo derrotado*

PruebaID	*PR-14*
Nombre	Subir de nivel al atacar y matar enemigo
Procedimiento	Atacar a un enemigo que pierde todos sus puntos de vida
Resultado esperado	Se da experiencia al jugador por el ataque realizado, se da experiencia a todos los jugadores vivos. Al subir de nivel sale la opción de aumentar 1 punto de atributo
Satisfactoria	
Requisitos relacionados	*RF-17, RF-18, RF-19*

*Tabla 61: Prueba **PR-14**: Subir de nivel al atacar y matar enemigo*

La realización de estas pruebas se efectúa en el apartado de Implementación y Evaluación marcando si cada prueba ha devuelto el resultado esperado y si ha seguido un procedimiento normal.

A pesar de tener un apartado dentro del formato tabular de las pruebas que especifica qué requisitos están involucrados dentro de cada una de las pruebas, se utiliza una matriz de trazabilidad para hacer un seguimiento de las correlaciones que existen entre pruebas y requisitos.

La matriz de trazabilidad se compone de filas de requisitos y columnas de pruebas del sistema donde se coloca una X en aquellas celdas donde se relacionen un requisito con una prueba.

	PR-01	PR-02	PR-03	PR-04	PR-05	PR-06	PR-07	PR-08	PR-09	PR-10	PR-11	PR-12	PR-13	PR-14
RF-01	X	X												
RF-02			X											
RF-03	X	X												
RF-04	X	X												
RF-05	X	X												
RF-06	X	X												
RF-07	X	X												
RF-08						X								
RF-09														
RF-10	X	X		X			X							
RF-11									X					
RF-12								X						
RF-13	X	X								X				
RF-14	X	X								X				
RF-15	X	X			X									
RF-16	X	X										X	X	
RF-17														X
RF-18														X
RF-19														X
RF-20	X	X												
RF-21													X	
RF-22												X		
RF-23	X	X									X			
RF-24												X		

Tabla 62: Matriz de trazabilidad: Pruebas y Requisitos

Como la jugabilidad se centra en gran medida en hacer elecciones, es decir, que los jugadores seleccionen una opción de las establecidas que puedan ejecutar, se utilizan números asociados a cada acción para hacerlo lo más fácil y directo posible para los jugadores. Por ello, la prueba *PR-01* que verifica ese *input* introducido por los jugadores alude a muchos requisitos.

4. DISEÑO

Este apartado del documento divide su objetivo en dos partes: la primera se centra en explicar la jugabilidad y los datos técnicos del juego para poder entender las mecánicas y decisiones de implementación tomadas. La segunda está dedicada a la explicación de la arquitectura y el diseño del sistema exponiendo sus partes y las conexiones existentes entre ellas.

Además, en la segunda parte, se utilizarán diagramas como los de clase para representar los componentes y las interacciones entre unos y otros.

4.1. DISEÑO DEL JUEGO

Las decisiones de diseño y mecánicas del juego son imprescindibles pulirlas antes de empezar a implementar ya que rigen toda la arquitectura y enfoque del proyecto. Por ello, este apartado se dedica a explicar de la forma más clara posible las diferentes entidades existentes, las mecánicas y los datos técnicos que estructuran al juego.

Entre ellos se hablará de la mecánica que sigue le juego para avanzar a través de una partida, la creación de los personajes, los distintos enemigos, los objetos que pueden usar los personajes, el sistema de experiencia y finalmente los dos núcleos de la jugabilidad: el movimiento por un mapa y el combate.

4.1.1. MECÁNICA DEL JUEGO

En este apartado se habla del progreso a través de una partida de forma poco detallada para obtener una visión general de cómo avanza el juego a medida que se realizan ciertas acciones. El juego funciona a través de consola y lo único que tienen que hacer es discutir las acciones conjuntas y escribir en consola la opción entre las planteadas.

Inicialmente, al ejecutar el juego, se pregunta a los jugadores cuántos van a jugar. Este número puede ser entre 1 y 3, incluidos. A partir de ahí se ejecuta en bucle la creación de cada uno de los personajes. Primero, se pregunta la clase del personaje, el nombre y los atributos iniciales. Luego, se elige el arma y, de forma conjunta, se elige la misión que se va a jugar.

Dependiendo de la misión elegida, se cuenta la historia detrás de ella, y se pide a los jugadores en conjunto que elijan la siguiente zona a la que se quieran mover. A partir de aquí, se sigue un bucle que empieza enfrentando a los jugadores en un combate por turnos contra el mismo número de enemigos que jugadores y acaba, si ganan el enfrentamiento, en un estado en el que pueden cambiar armas, abrir cofres y recoger nuevo inventario. De ahí, eligen nueva zona para moverse y se repite el proceso hasta que alcanzan la zona final. Dependiendo de la misión, se mostrará la información correspondiente a los jugadores y se terminará con un enfrentamiento final.

4.1.2. PERSONAJES Y ENEMIGOS

Es necesario juntar tanto personajes como enemigos en un mismo apartado ya que existe una relación esencial de la jugabilidad existente entre ellos.

Cada jugador tiene en su control a un personaje y puede crearlo a su manera modificando determinados aspectos que lo caracterizan.

Todo personaje tiene unos puntos de vida y 3 **atributos** que le proporcionan beneficios en el combate. Los puntos de vida dependen de uno de los atributos, se pueden recuperar con pociones y si llegan a 0 ese personaje muere y no puede jugar más. Estos atributos mencionados son: fuerza, resistencia y vitalidad. La **Fuerza** determina el daño adicional que realiza el personaje, la **Resistencia** determina la reducción de daño y la **Vitalidad** los puntos de vida máximos que puede tener el personaje. Las fórmulas que determinan sus efectos son:

- **Fuerza**: $Daño\ adicional = floor(\frac{Fuerza-3}{4})$
- **Resistencia:** $Reducción\ de\ daño = floor(\frac{Resistencia+1}{5})$
- **Vitalidad**: $Puntos\ de\ vida\ máximos = 10 + floor(\frac{Vitalidad-2}{2})$

**floor: Función que aproxima un decimal hacia valor más bajo (Ejemplo: floor(8.4)=8)*

La característica más importante es la **clase** del personaje. Esta elección modificará los atributos iniciales con los que el personaje empieza y también influirá en el daño realizado a determinados enemigos y el daño recibido de determinados enemigos. Cada personaje puede pertenecer a una Escuela que coincide con el tipo de clase siendo las opciones: Escuela del Oso, Escuela del Grifo y Escuela del Lobo.

	Atributos iniciales		
Escuela	***Fuerza***	***Resistencia***	***Vitalidad***
Oso	6	10	6
Grifo	6	6	10
Lobo	10	6	6

Tabla 63: Atributos iniciales en función de la Escuela

Los valores máximos que pueden alcanzar en los atributos son 20, 15 y 20 respectivamente para Fuerza, Resistencia y Vitalidad; se puede seguir invirtiendo puntos en ellos, pero no se aporta más beneficio. Por ello, la vida máxima más alta que se puede alcanzar es 19; la reducción de daño más alta es de 3 puntos y el daño extra mayor es 4.

Por otro lado, los **enemigos** no poseen atributos y sus puntos de vida iniciales son fijos con un valor de 14. Además, cada enemigo es de un tipo diferente dentro de 6 opciones:

espectro, bestia, necrófago, híbrido, humano e insectoide; y cada tipo contiene un subtipo que solo aporta más variedad al juego, pero no otorga diferentes cualidades. Por ejemplo, si un enemigo es de tipo espectro puede tratarse de una *Banshee, Doncella de la Peste* o *Etéreo.* Dependiendo del tipo de enemigo, el daño que realizan a un personaje con una determinada clase varía, pero el daño base del que parten es el mismo: 8.

	Espectro	***Bestia***	***Necrófago***	***Híbrido***	***Humano***	***Insectoide***
Escuela del Oso						
Escuela del Grifo						
Escuela del Lobo						

Tabla 64: Modificaciones de daño según clase y enemigo

En caso de atacar a un enemigo, si es una celda verde, aumenta en 1 el daño que se realiza contra ese enemigo, pero si es roja, se reduce en 1 el daño que se realiza.

Por otro lado, si un enemigo ataca a un personaje y la celda coincidente es verde, se reduce en 1 el daño que se reciba mientras que, si la celda es roja, aumenta en 1 el daño que se reciba.

4.1.3. INVENTARIO

Cada personaje está equipado con un arma y con pociones que podrá usar en combate. Estos elementos ayudan al jugador a completar el objetivo del juego de forma más sencilla, aunque depende de la utilidad que se da a ambos objetos.

Todas las **armas** tienen un daño base a partir del cual se calculará el daño que efectúa el jugador a un enemigo. Ese daño base se verá afectado por atributos como la Fuerza y por la relación clase-tipo de enemigo de los enfrentados. El rango de daño base que tienen todas las armas está entre 2 y 6 habiendo menos cantidad de armas de daño alto que de daño bajo. Estas armas provienen de una lista previamente construida y se generan en dos fuentes: en la creación de personajes cuando se pide al jugador elegir entre 3 armas aleatorias y cuando se abre un cofre, en donde solo se genera una.

Además del daño base, cada arma tiene características propias que proporcionan beneficios al jugador que la posee. Estas características se dividen en daño adicional a ciertos tipos de enemigos y aumento de una cantidad determinada de un atributo. Por ende, al daño base previamente mencionado se le debe añadir estas características.

Las **pociones**, por otro lado, no son permanentes y tienen un único uso cada una. Cualquier jugador cuyos puntos de vida estén por debajo de los puntos máximos de vida calculados por su Vitalidad puede hacer uso de una poción para reestablecer hasta 5 puntos de vida sin superar el valor de vida máxima. El jugador puede usar la poción sólo cuando esté en combate y cuando sea su turno, además no podrá atacar ese turno ya que el uso de la poción habrá gastado el turno.

4.1.4. MOVIMIENTO POR EL MAPA

El movimiento de los jugadores se rige por el mapa en el que empiezan la misión, por lo que la estructura del mapa está determinada por la elección tomada al final de la creación de personajes. Las misiones existentes son dos, una corta y otra más larga, para que los jugadores tengan elección al menos entre dos en función de la duración de la partida que deseen.

El movimiento es solo en un sentido, aunque siempre se presentan entre 1 y 3 alternativas para cada movimiento con el objetivo de evitar un "avance de túnel" en donde los jugadores no tienen más remedio que seguir el camino que el juego plantea. Como no se puede retroceder y no todas las zonas están conectadas con todas, no se puede alcanzar cada una de las zonas de ninguno de los dos mapas.

Las localizaciones de los puntos del mapa en las misiones están basadas en un mapa real de las novelas de *Andrzej Sapkowski*.

4.1.5. COMBATE

El sistema de combate está basado en un sistema de turnos que sigue el mismo procedimiento que los juegos por turnos explicados en el apartado 2.1.1. del documento.

Los primeros en atacar serán los jugadores, siempre en un orden determinado en función del orden seguido en la creación de los personajes. Después, atacarán los enemigos en un orden específico que va en función de la generación de dichos enemigos.

Los jugadores en cada turno van a tener hasta 2 posibles acciones: atacar o tomarse una poción; en caso de tener toda la vida, solo se tendrá una opción. Al atacar, el jugador que esté actuando ese turno podrá elegir a qué enemigo va a atacar realizando un valor de daño que reducirá la vida de ese enemigo. Por otro lado, al utilizar una poción, el personaje cura sus puntos de vida hasta 5 sin superar la vida máxima y no puede ejecutar ninguna acción más ese turno.

En caso de perder todos los puntos de vida, el jugador cuyo personaje haya sido derrotado no podrá jugar más.

4.1.6. SISTEMA DE EXPERIENCIA

Cada personaje de la partida tiene un nivel y una barra de experiencia que constituyen el sistema de experiencia del juego. Inicialmente, todos los personajes empiezan a nivel 1 y según van obteniendo experiencia, se va subiendo de nivel.

Hay dos formas de obtener experiencia y ambas se consiguen en combate: realizando daño a enemigos y derrotándolos. Cuando un personaje ataca a un enemigo, obtiene la misma cantidad de experiencia que el daño efectuado y si, además, se mata a ese enemigo todos los jugadores vivos reciben un extra de 20 puntos de experiencia. Cuando se alcanza

un valor determinado de experiencia, el personaje sube de nivel y su barra de experiencia vuelve a estar vacía o a tener tantos puntos como los que sobraron al subir de nivel.

Al subir de nivel, al jugador se le pide elegir subir un punto de los tres atributos explicados en el apartado 4.1.2. Además, al subir de nivel, el valor máximo de puntos de experiencia que deben conseguir es mayor. Más específicamente, empezando por 50, aumenta de 10 en 10 cada vez que se sube de nivel.

4.1.7. TEMÁTICA DEL JUEGO

Dentro de las posibles ambientaciones y temáticas que se pueden utilizar dentro de un juego de rol se tiene una variedad abismal partiendo de escenarios medievales hasta llegar a ambientaciones espaciales y futuristas.

En este caso, se ha utilizado un ambiente medieval, pero utilizando aspectos eslavos procedentes del folklore y de las tradiciones procedentes de la cultura europea del este. Los elementos del proyecto que han bebido más de este folklore son las criaturas fantásticas utilizadas para enfrentar a los jugadores.

Los 6 tipos de enemigos creados para crear la mecánica de combate de contrincantes fuertes o débiles dependiendo de la clase elegida, son extraídos a partir de historias y leyendas de criaturas que habitan en ellas. En sí, los tipos de enemigos son muy genéricos, pero al profundizar en el subtipo se aprecia más las referencias a este folklore. Este subtipo, es importante recordar, no aporta nada en las mecánicas del juego, sino que aumentan la variedad de enemigos para evitar la monotonía y repetición de los combates.

Sin embargo, el origen cultural de estas criaturas puede ser confuso e incluso provenir de varias fuentes que no tienen mucha relación entre ellas. Como es el caso del *Ghoul* cuyo origen se asocia a la cultura pre-Islámica. Esta criatura es un demonio que habita en lugares inhabitados y se alimentan de cadáveres [37]. A medida que esta historia se desplaza al oeste, hacia Europa, se varían varios aspectos de esta criatura.

Otro ejemplo de diferente origen es el caso de las *Banshee*, espíritus de mujeres que vaticinan a una persona la muerte de un pariente cercano. Su origen se remonta al folklore irlandés y se ha transmitido hasta muchas partes de Europa [38].

En este proyecto, todas las criaturas mencionadas se han modificado en cierto modo para que se pueda producir un enfrentamiento con los jugadores. Por ejemplo, según las historias de las *Banshee*, no son agresivas e incluso ayudan a la gente.

Además del folklore y las tradiciones, se han utilizado elementos de la saga fantástica de *Geralt de Rivia* escrita por el autor *Andrzej Sapkowski*. De estas obras se ha utilizado la ambientación como una época cercana a la edad media, un par de criaturas invento del autor, los personajes referidos a las clases y las localizaciones dentro de cada uno de los mapas de las misiones.

4.2. ARQUITECTURA DEL SISTEMA

El sistema posee elementos que establecen conexiones entre ellos y transmiten información para realizar las determinadas tareas que tienen como objetivo el progreso del programa. Estos elementos, señalados en la figura 3, componen la arquitectura del sistema donde cada uno tiene sus funcionalidades y restricciones.

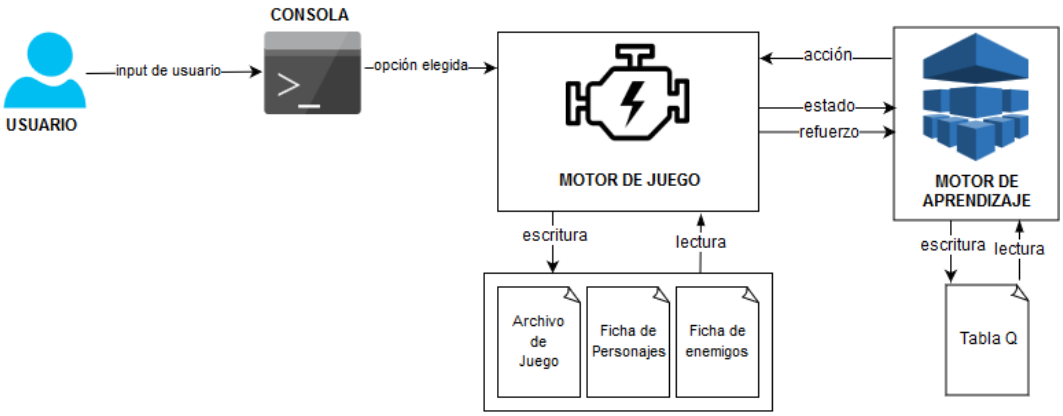

Figura 3: Arquitectura del sistema

El componente más extenso de la arquitectura y el que más funcionalidades posee es el Motor de Juego. Este elemento contiene todo el código responsable del avance de los jugadores por el juego donde se incluyen los apartados de creación de personajes, avance por el mapa, estructura del combate y demás aspectos más secundarios. Todas las interacciones y datos importantes para los jugadores se muestran mediante el uso de la consola.

Por otro lado, se tiene el motor dedicado al aprendizaje por refuerzo del agente automático. Esta sección de la arquitectura se dedica al entrenamiento previo del agente obteniendo un archivo que contiene la tabla Q, donde retiene información para su siguiente funcionalidad: la respuesta de los *NPC* en las situaciones de combate. Aquí, además, también se posee la fuente de la respuesta del agente humano: un archivo de estados con la acción correspondiente.

Además, se tienen determinados archivos que establecen la situación del juego en cada momento. El archivo que se mantiene existente toda la partida contiene la información más relevante como en qué sitio están los jugadores, qué armas tienen, los atributos de éstos, las zonas conectadas entre sí… En el caso del combate, se generan dos archivos que contienen, primero, los datos de los enemigos que han sido generados de forma aleatoria en la zona nueva visitada y, segundo, los datos de los jugadores con datos como sus puntos de vida y los que quedan vivos.

Conociendo los componentes se pueden establecer las conexiones que existen entre ellos y más adelante el flujo principal que se sigue en una partida normal.

El Motor, al ser el elemento principal de todo el juego, está comunicado con todos los elementos secundarios. En repetidas ocasiones a lo largo de la partida, el Motor lee y escribe los archivos de juego y de combate en determinados momentos de la partida. Además, el Motor actúa como entorno dentro del proceso de aprendizaje por refuerzo otorgando el estado nuevo después de recibir la acción realizada por el agente automático.

Y, lo más importante, se comunica directamente con los jugadores mediante el uso de la consola.

Antes del proceso seguido de forma corriente por el juego, se debe analizar el proceso de aprendizaje enfocándolo a la arquitectura del sistema. Para ello se requiere un Motor de juego modificado que esté focalizado exclusivamente en el combate y que también actúe de entorno para el aprendizaje. El proceso es más sencillo: por cada sesión el Motor modificado informa del estado al motor de aprendizaje; éste efectúa los cálculos necesarios y hace una consulta al archivo de la tabla Q, de la cual lee y en la cual luego escribe, devolviendo al Motor modificado la acción que quiere tomar. Aquí se hace una consulta al archivo de estados del agente que se quiera utilizar y se devuelve la acción del estado que coincida. Este proceso se repite hasta que un bando es derrotado y se repite todo tantas veces como sesiones se hayan establecido.

El proceso de conexiones entre los elementos dentro de una partida empieza en los jugadores. El Motor pregunta a los jugadores por consola cuántos van a ser y, en función de ese valor, se establece un bucle de interacciones entre el Motor y dichos jugadores, donde el Motor establece la información que quiere que responda el usuario y plantea las opciones disponibles mientras que los jugadores responden con las opciones establecidas. Este proceso se va a repetir en diferentes partes del juego, por lo que se referirá a ello con el término "petición-respuesta" del Motor y los jugadores.

Tras tener los personajes creados, se genera el archivo del juego con los datos de los jugadores y de la estructuración del juego como las zonas disponibles y las conexiones entre ellas. Después de escribir el archivo, se vuelve a hacer una petición-respuesta para saber a qué zona moverse leyendo el archivo y mostrando las opciones disponibles desde la zona en la que están los usuarios. Al moverse a una nueva zona, se modifica el archivo del juego para actualizar las posiciones de los jugadores y se crean los dos archivos de combate: uno que lleve la información de los jugadores y otro que contenga la información de los enemigos generados aleatoriamente en la nueva zona.

El combate es el proceso que tiene mayor número de conexiones y transiciones a través de la arquitectura del sistema. Como empieza el usuario, se realiza una petición-respuesta del Motor para preguntar qué acción elige efectuar y a qué *NPC* quiere atacar. Después de efectuar las peticiones-respuestas de todos los jugadores, habrá acabado su turno y se debe pasar al turno de los *NPC*. Pero antes, hay que modificar el archivo de los personajes en caso de haber utilizado pociones y el archivo de los *NPC* con el daño causado por los jugadores, e incluso eliminar aquellos que hayan sido derrotados. Para que los *NPC* puedan actuar, el sistema, que funciona como entorno, transmite el estado del juego estructuralmente establecido en el apartado de Aprendizaje por Refuerzo del agente. Tras conocer el estado en el que se encuentra, el motor de aprendizaje hace una lectura al archivo que contiene la tabla Q para poder elegir qué acción es la que va a ejecutar. Este proceso se repite dentro de un turno tantas veces como *NPC* haya con vida. Tras el turno de los enemigos, se modifica el archivo de los personajes con respecto al daño causado por los *NPC* y, en caso de haber personajes con vida, se vuelve al turno de los jugadores, empezando el ciclo de combate de nuevo. Este proceso acaba cuando uno de los bandos está completamente eliminado. Es importante que la escritura de ambos archivos de combate se haga al terminar el turno de uno de los bandos y no antes.

En caso de haber ganado el combate los jugadores, se vuelven a efectuar peticiones-respuesta que modifican el archivo del juego en los aspectos referentes a las armas equipadas y las pociones que tiene cada uno. Y se repite el bucle de movimiento y combate.

4.3. INTERFAZ DE USUARIO

Al haber centrado el proyecto en un juego donde los *NPC* no actúan mediante un sistema simple, la sección gráfica del juego está en sus puntos menos desarrollados. Como se tiene el esqueleto del proceso de juego, los jugadores podrán acceder a dicho juego mediante la consola de Python.

Los jugadores solo pueden ejecutar un número de acciones muy limitada. Para poder facilitar tanto el proceso de implementación como la jugabilidad de los usuarios, el sistema cada vez que necesite *input* de los jugadores, plantea la información que necesita y se presentan entre 1 y 3 opciones a la que se les asocia un número. Número que tendrán que elegir los usuarios para realizar esa acción correspondiente.

Por tanto, la interfaz está del mismo formato que la consola sin ningún tipo de representación gráfica o fuera del formato textual como se muestra en la figura 4.

```
Jugador  1 , ¿qué personaje quieres elegir?
1. Brujo de la Escuela del Oso
2. Brujo de la Escuela del Grifo
3. Brujo de la Escuela del Lobo

Selecciona 1, 2 ó 3:
1
```

Figura 4: Ejemplo de elección de opciones en consola

Conociendo las opciones que pueden plantear y las acciones que los jugadores pueden ejecutar, se puede construir un esquema gráfico que muestre la evolución de los jugadores a través de una partida normal. Se utiliza un esquema de cubos y flechas para representar el avance por la partida.

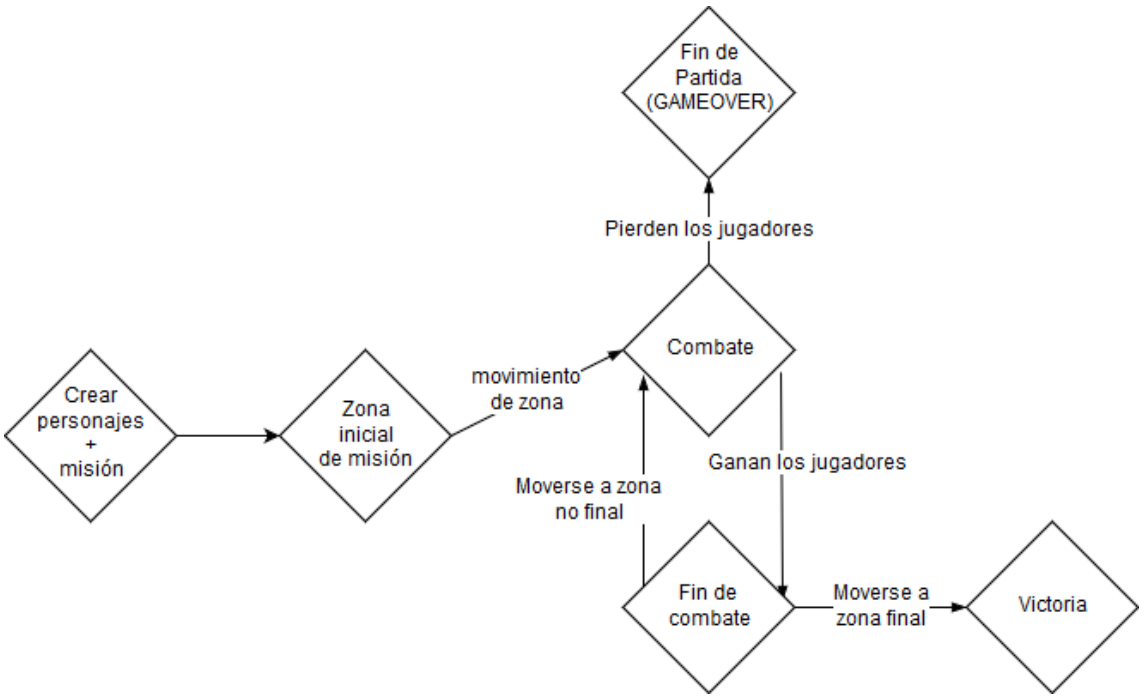

Figura 5: Diagrama de flujo del juego

Según la figura 5, este flujo representa los estados en los que pueden estar los jugadores en cada momento y las transiciones que hacen moverse de un estado a otro. Como se puede ver, el bucle principal que compone la jugabilidad de la partida se centra en las transiciones de combate a fin de combate, y viceversa, donde sólo se sale de ese bucle si se pierde la partida o se llega al estado final, es decir, si se gana la partida.

4.4. DIAGRAMAS DE COMPONENTES

Partiendo de la arquitectura recientemente expuesta en el documento, se puede entrar en detalle en cada una de las partes para la creación de los diagramas de componentes. Estos diagramas muestran cómo está organizado el sistema y qué conexiones existen entre los componentes que hay en él como se aprecia en la figura 6.

Los componentes primarios que componen al sistema son el Motor de juego, la consola y el motor de aprendizaje. Dentro de cada uno de ellos se encuentran otros componentes más secundarios cuya funcionalidad está marcada por el componente primario al que pertenecen.

El Motor de juego, en la figura 7, se comunica con el motor de aprendizaje informando del estado en el que se encuentra la partida en cada momento y a su vez el motor de aprendizaje se comunica con el Motor de juego para devolver la acción que el agente realiza.

A su vez, tanto el Motor como el motor de aprendizaje se comunican con la consola para transmitir información relevante a los jugadores. El Motor de juego, por un lado, transmite a la consola la información que al jugador le tiene que resultar útil para tomar

una decisión y, por otro lado, el motor de aprendizaje transmite a la consola la acción tomada por el agente automático.

Dentro del Motor de juego, tenemos unos archivos junto con un gestor que hace las consultas apropiadas a los archivos apropiados para informar de la situación de la partida o para actualizar la información que se tiene.

Profundizando en el motor de aprendizaje, como en la figura 8, se tiene un procedimiento similar al del Motor de Juego, donde el modelo del agente automático ya entrenado hace consultas al archivo que contiene la tabla Q. Esto le permite conocer qué acción va a ejecutar en función del estado transmitido como entrada al componente del motor de aprendizaje. Devolviendo así dicha acción de vuelta al Motor de juego y a la consola para informar a los jugadores.

Todos los casos de uso del usuario se establecen como 'Petición de usuario' para generalizar las interacciones de los jugadores con el sistema y no utilizar varios diagramas que aludan a un mismo procedimiento común.

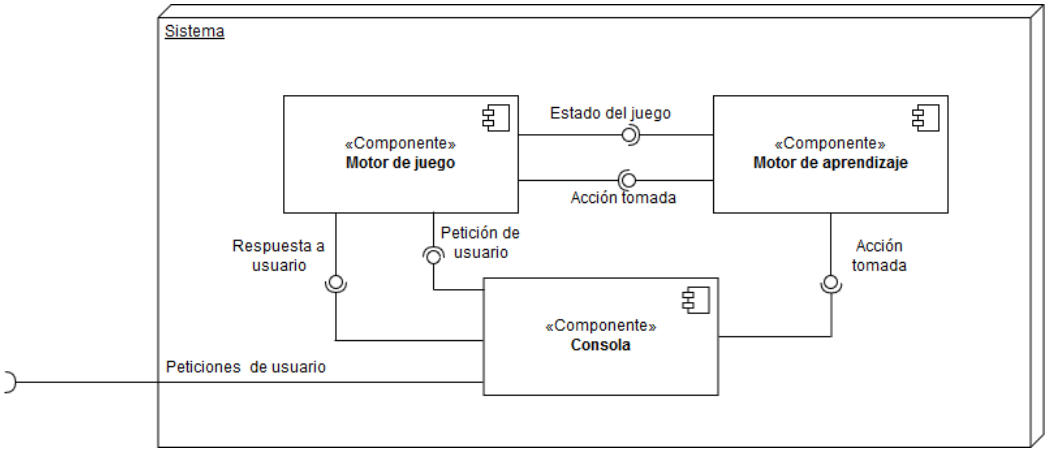

Figura 6: Diagrama de Componentes: ***Sistema***

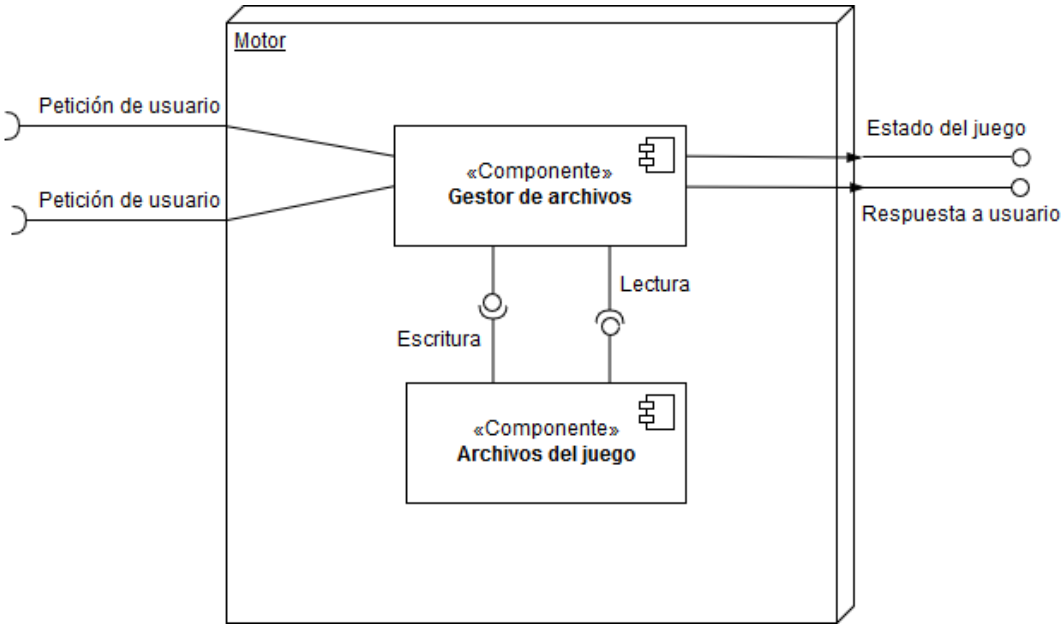

Figura 7: Diagrama de Componentes: ***Motor***

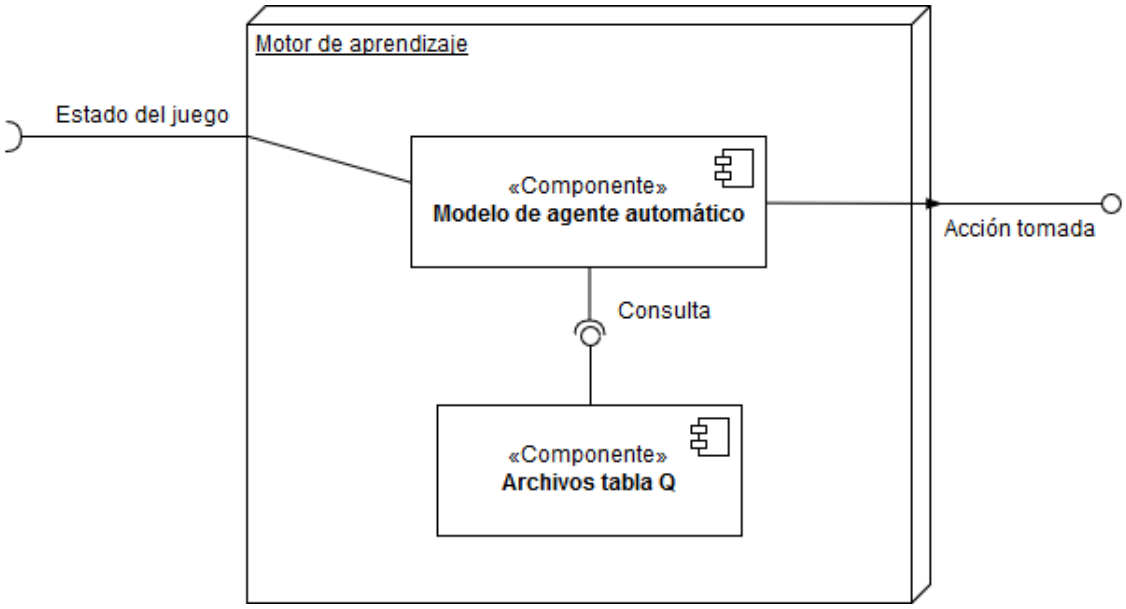

Figura 8: Diagrama de Componentes: ***Motor de Aprendizaje***

4.5. DIAGRAMAS DE CLASE

Los diagramas de clase representan las diferentes clases de elementos que existen en el sistema, tal y como se muestra en la figura 9. Cada diagrama hará alusión a una clase en particular, marcando cada uno de los atributos que la componen junto con las diferentes funciones que se asocian a ella. Además, se exponen las relaciones entre las clases.

La primera clase contiene la información de cada **personaje** que le sirve al sistema para realizar los cálculos necesarios en el combate al igual que acciones secundarias fuera del enfrentamiento contra el agente automático. La siguiente alude a los **enemigos** generados en el sistema que sirve para los cálculos en el combate. Luego, la clase de **motor** que controla el avance a través del juego y comunica con todas las demás clases. Y finalmente, la clase de **zona** que controla qué elementos se generan junto con los enemigos relacionándola con la segunda clase.

La funcionalidad de cada una de las funciones y métodos que tienen las clases se explicarán en más detalle en el apartado de los diagramas de secuencia. Sin embargo, como los nombres utilizados para referirse a estas funciones son lo suficientemente explicativos, solo faltaría entrar en el detalle de qué conectan y con qué. Esto también se hace en el siguiente apartado.

Las funciones más importantes del motor son las escrituras del archivo ya que la actualización del juego es la parte que controla el avance en la misma partida.

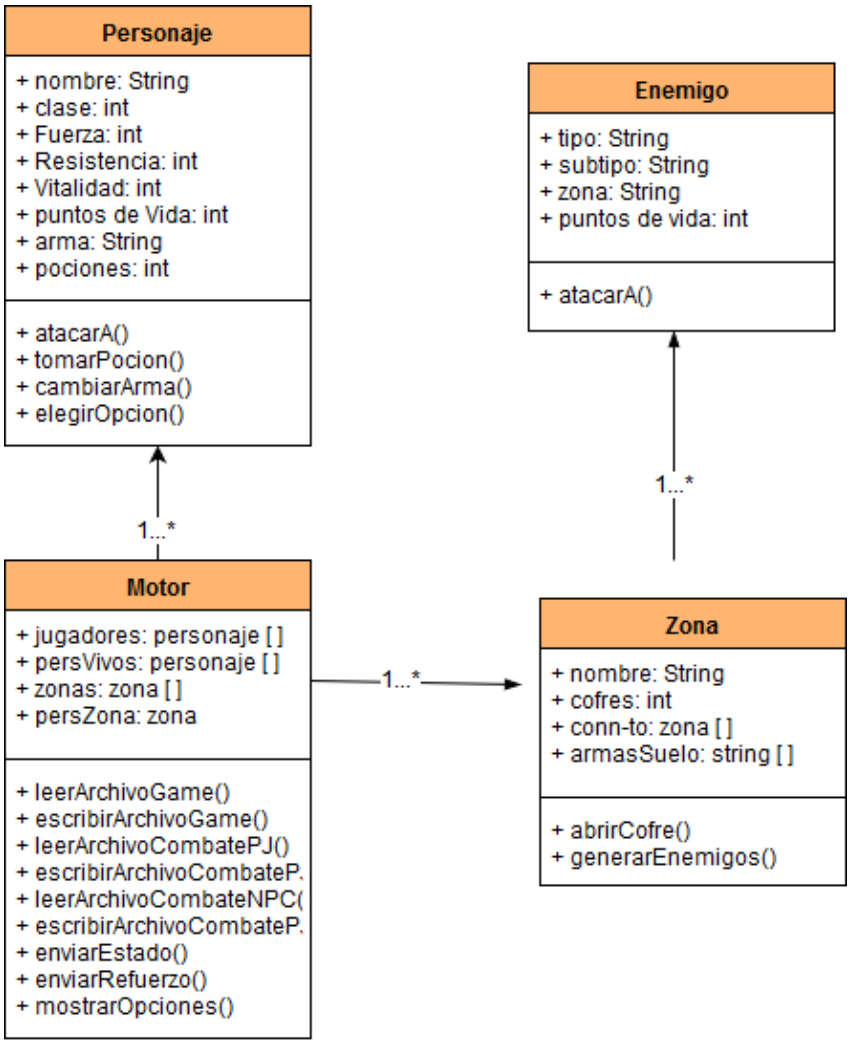

Figura 9: Diagrama de Clases

4.6. DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia tienen como objetivo conectar los diagramas de clase y los casos de uso previamente establecidos para conseguir la exposición de las interacciones que hay entre los objetos relacionados y los eventos que ocurren.

A continuación, se muestran cada uno de los casos de uso disponibles en el sistema utilizando conexiones entre las distintas clases.

Los seis primeros casos de uso, mostrados en la figura 10, se centran en un mismo procedimiento que es común para todos ellos. Por lo que el diagrama se comparte para los seis. La idea es que al jugador se le planteen varias opciones de las cuales tendrá que elegir una entre todas.

Muchas funciones y métodos de las clases preestablecidas no se presentan en estos casos de uso ya que son acciones que se efectúan ajenas a las funcionalidades que tienen los jugadores.

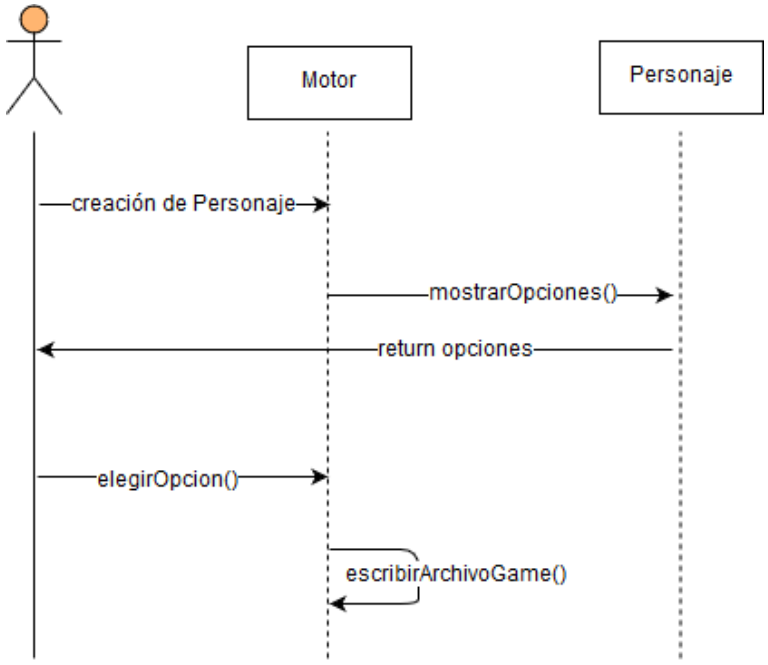

Figura 10: Diagrama de Secuencia: ***Elección de opciones***

El siguiente caso de uso tiene el mismo procedimiento, pero la petición del usuario es moverse de mapa en lugar de crear los personajes, como se muestra en la figura 11. Sin embargo, también tiene otra funcionalidad aparte de la petición de un *input* de usuario.

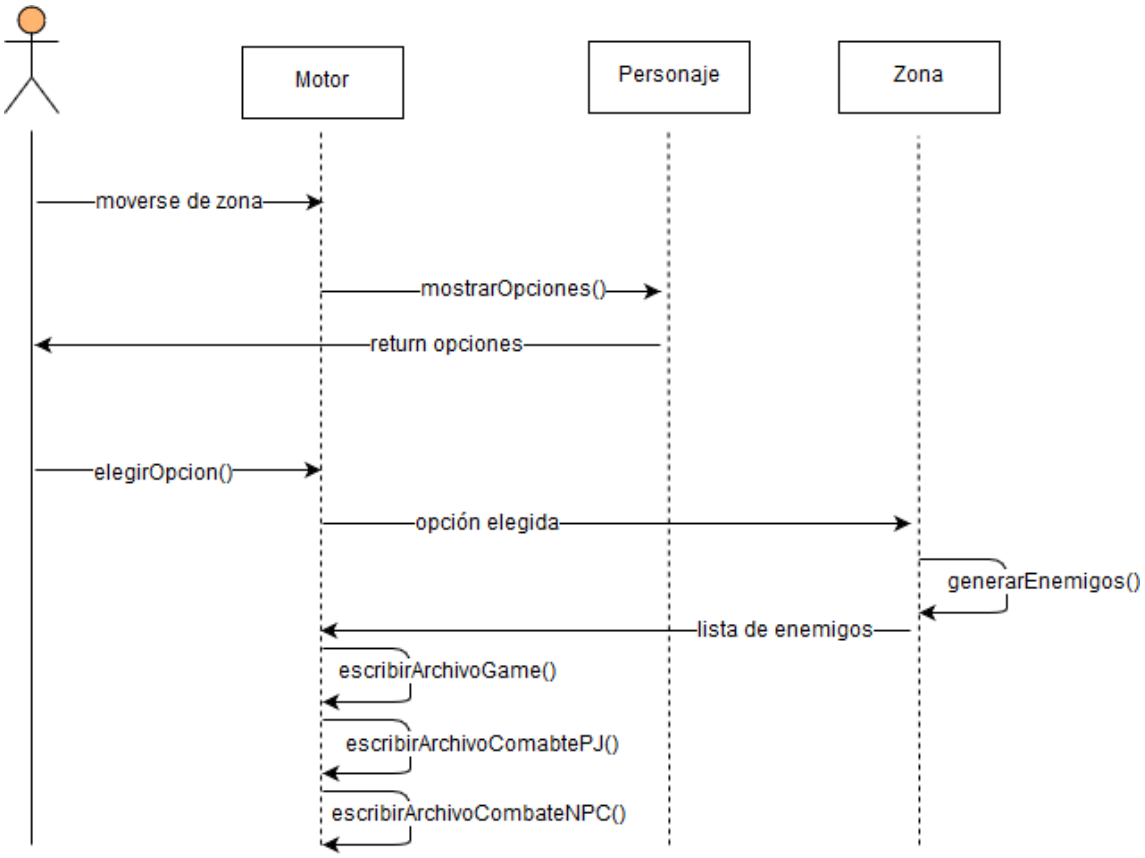

Figura 11: Diagrama de Secuencia: ***Moverse por el mapa***

El siguiente caso de uso, mostrado en la figura 12, se centra en el ataque de un jugador a un enemigo, por el cual necesitaremos la clase personaje que aplique el método de atacar a la clase enemigo especificado y luego la interacción del motor para actualizar los archivos.

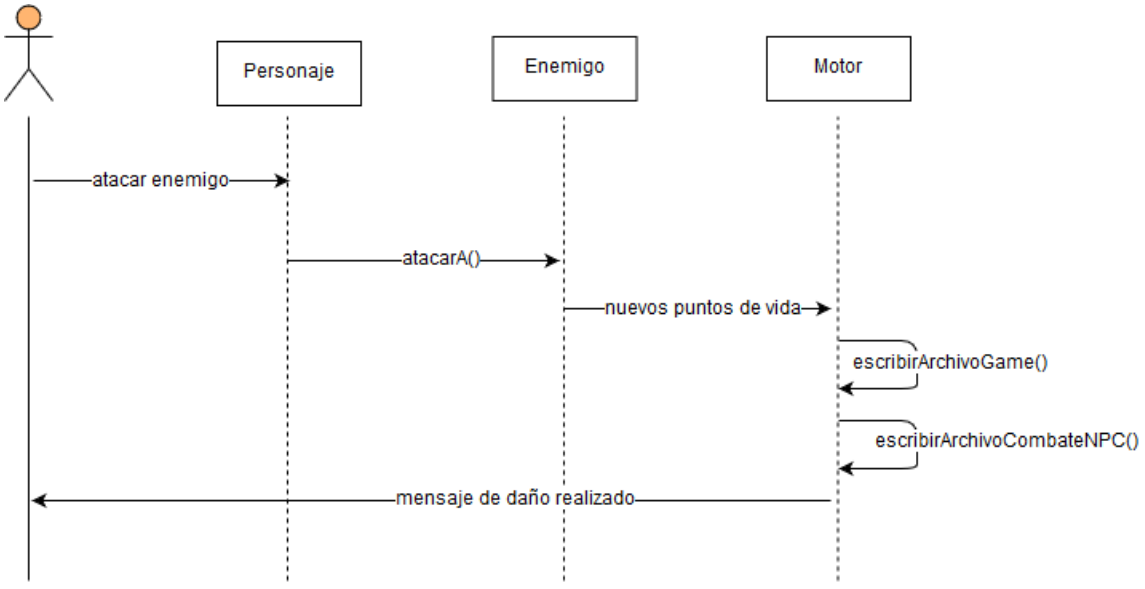

Figura 12: Diagrama de Secuencia: ***Atacar a enemigo***

Tomarse una poción, como caso de uso de la figura 13, solo requiere aludir al personaje para efectuar ese método específico que se toma la poción y luego informar al motor para actualizar los archivos.

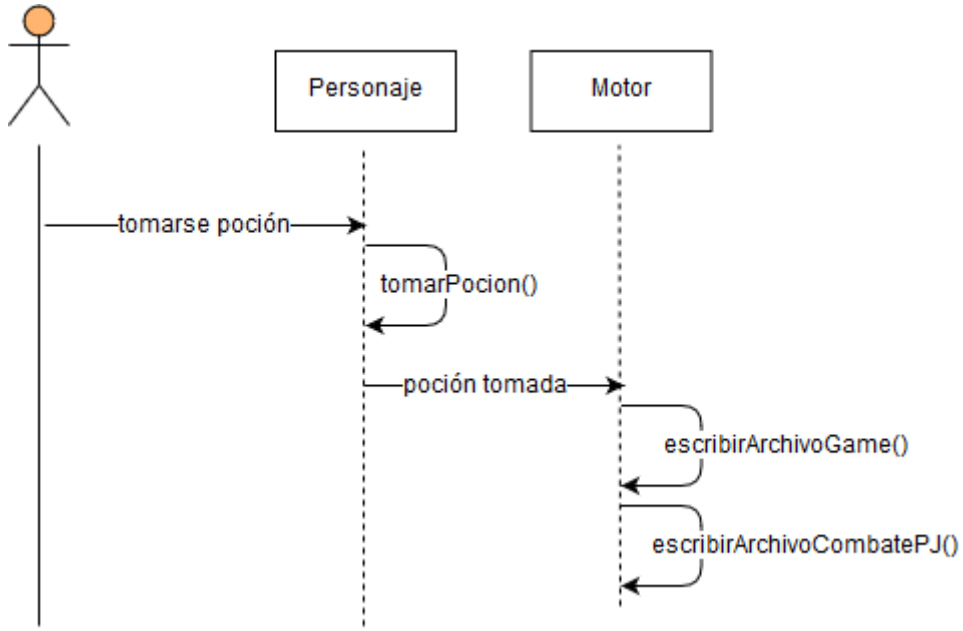

Figura 13:Diagrama de Secuencia: ***Tomar poción***

Para abrir un cofre, en el caso de uso de la figura 14, se requiere aludir a la clase zona para efectuar esa acción mediante la función que tiene asociada e informar al motor para modificar el archivo del juego.

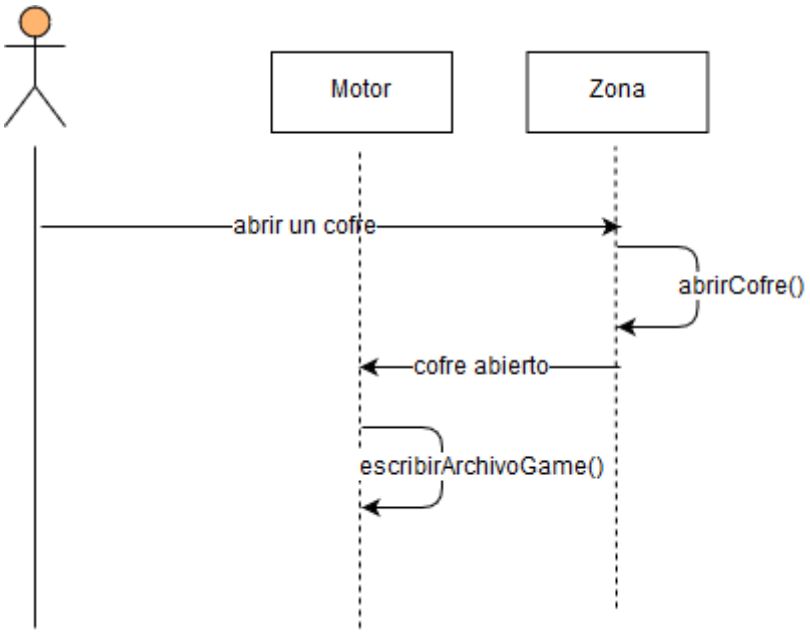

Figura 14: Diagrama de Secuencia: ***Abrir cofre***

Los 3 siguientes casos de uso tienen una relación muy común por lo que se los tratará en un mismo diagrama y en la figura 15. Los casos 11 y 13 realizan el mismo procedimiento, pero desde distintas fuentes mientras que el caso 12 es la consecuencia de ambas.

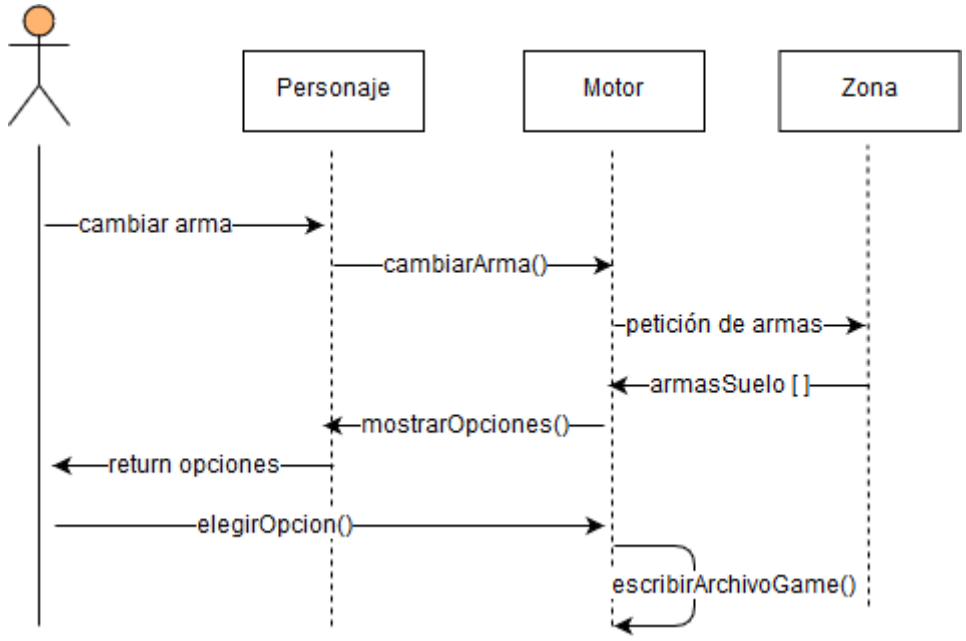

Figura 15: Diagrama de Secuencia: ***Cambiar de armas***

Para el caso de uso que se centra en que el personaje coja una poción y se la equipe se rige por el mismo principio que el diagrama anterior. Y, por último, la elección de subir un punto de atributo por subir de nivel como último caso de uso, sigue el mismo procedimiento que el primer diagrama de secuencia.

5. IMPLEMENTACIÓN Y EVALUACIÓN

Esta sección del documento se centra en la implementación del proyecto y las evaluaciones realizadas para medir la calidad del trabajo y de los distintos parámetros a establecer.

Inicialmente, se analiza el proceso estructurado para el aprendizaje del agente automático encargado de enfrentarse a los jugadores en combate. Partiendo de la estructuración de los estados, el entrenamiento y acabando con la evaluación de todos los casos utilizados en las pruebas de entrenamiento y validación.

Más adelante, se muestran los resultados de todos los tipos de prueba realizadas, tanto pruebas del sistema como pruebas de usuario. Además, se expone la estructuración de dichas pruebas al igual que el enfoque de cada una de ellas.

5.1. APRENDIZAJE DEL SISTEMA

Al tratarse de un sistema basado en aprendizaje por refuerzo y que trabaja con el algoritmo *Q-Learning*, es necesario fijar los parámetros que caracterizan al aprendizaje y preparar los distintos entornos donde se realizarán los entrenamientos y las pruebas.

Es importante mencionar que este aprendizaje por refuerzo solo se utiliza cuando el número de jugadores es de 2 ya que si se utilizase para 3 el conjunto de estados sería excesivo y si es solo un jugador, el agente solo tiene una elección y no se necesita ningún política inteligente.

5.1.1. ESTRUCTURACIÓN DEL APRENDIZAJE

Como ya se ha estudiado en el apartado 2.2.3. del documento, el aprendizaje por refuerzo está compuesto por 3 elementos fundamentales que se van a comunicar entre agente y entorno: estados, acciones y refuerzos.

El **estado** marca las condiciones y atmósfera en la que se encuentra el agente en cada momento estableciendo las posibles acciones que puede ejecutar el agente. Por ello, se deben utilizar elementos del juego para utilizarlos como atributos del estado. El objetivo de esta construcción es que el agente, que representa a los enemigos o *NPC*, conozca cierta información de ambos jugadores para actuar en función de los valores de esos datos que percibe. Pero no se le da toda la información.

El estado está formado por: $< R_{H1}, R_{H2}, C_{H1}, C_{H2}, HP_{H1}, HP_{H2}, HP_{NPC}, Type >$ siendo cada uno de los elementos:

- **R_{H1} :** Resistencia del Personaje 1
- **R_{H2} :** Resistencia del Personaje 2
- **C_{H1} :** Clase del Personaje 1
- **C_{H2} :** Clase del Personaje 2
- **HP_{H1} :** Puntos de vida del Personaje 1

- **HP_{H2} :** Puntos de vida del Personaje 2
- **HP_{NPC} :** Puntos de vida del agente (*NPC*)
- **Type :** Tipo de enemigo

Una opción que se tuvo en cuenta para hacer la construcción de los estados fue la separación de distintas tablas Q para los distintos tipos de enemigos que había, omitiendo este atributo del estado, pero, aunque los estados fuesen menores en cada una de esas tablas, el entrenamiento se alargaría demasiado. Por ello, se descartó la idea para sustituirla con la presentada.

Para saber cuántos estados vamos a tener debemos estimar qué posibles valores van a tener cada uno de los atributos del estado. En el caso de la clase de personaje o el tipo de enemigo, el valor es fijo siendo 3 para las clases y 6 para el tipo de enemigo. Sin embargo, para los puntos de resistencia y los puntos de vida se tienen muchos más valores posibles y si utilizamos cada uno de esos valores, el número de estados sería excesivo ya que la combinación de valores de atributos se multiplicaría. Por ello, se hacen agrupaciones de los valores, pero agrupaciones lógicas para intentar mantener una coherencia de los valores agrupados. Por ejemplo, para los puntos de vida, un grupo fijo es el caso de 0 ya que implica que ha sido derrotado o, por ejemplo, agrupar los valores de vida por los que cualquier golpe de un contrario derrote a ese personaje / *NPC*.

La resistencia está agrupada en 3 grupos en función de la reducción de daño que tenga: el primero 6-8, el segundo 9-13, y el último 14-15 siendo las reducciones de 1, 2 y 3 puntos respectivamente.

La vida del *NPC* se divide en 4 grupos: empezando por 0, seguido de 1 a 5 donde cualquier golpe de un personaje puede acabar con él, de 6 a 10 y ya el último de 10 a 14.

En caso de la vida de los personajes, se utilizan 5 grupos: también empezando por 0, seguido de 1 a 3 siendo los valores con los que con un único ataque de un *NPC* el personaje muere cualquier caso, de 4 a 8 donde puede que muera si le atacan, pero no siempre, luego de 9 a 13 y finalmente de 14 a 19 siendo 19 el valor máximo que puede tener un personaje como vida máxima.

Por cada estado, el agente podrá efectuar dos **acciones**: atacar al Personaje 1 o atacar al Personaje 2. Aunque el personaje tenga 0 puntos de vida, el agente podrá seguir atacándolo, aunque esto supone una repercusión negativa en el refuerzo recibido.

El **refuerzo** lo recibirán los agentes, cada uno el suyo, después de que los jugadores en el turno siguiente hayan ejecutado sus acciones. Esto facilita a los agentes recibir refuerzo en función no solo de sus acciones sino de las acciones consecuentes a las suyas**.** El refuerzo va a depender de las situaciones en las que se encuentre el agente y la respuesta que tenga ante ellas. Si el agente ataca a un jugador, el refuerzo acumulado aumenta tanto como el daño realizado a ese jugador ya que disminuir los puntos de vida de los jugadores es una acción apropiada. Si, además, el personaje muere, recibirán un extra de 100 puntos de refuerzo. Pero si, por otro lado, atacan a un personaje que ya ha muerto o muere uno de los *NPC* al ejecutar una determinada acción, el agente debe aprender que no es una acción beneficiosa y se le otorgará un refuerzo negativo de -200.

5.1.2. AGENTE HUMANO

Como ya se menciona en el apartado 3.2.1.1 del documento, la creación de un agente automático que simule los movimientos de un jugador es necesario para poder realizar un entrenamiento automático sin la necesidad de un usuario que actúe de forma manual en el proceso.

Este agente humano debe analizar el estado en el que se encuentra y devolver la acción a ejecutar, es decir, atacar al enemigo 1 o al enemigo 2. Por tanto, la construcción del estado 'humano' debe contener todos los posibles elementos/atributos relevantes que no necesariamente tienen que coincidir con los atributos de los estados del agente de aprendizaje automático. Por ejemplo, para el agente humano la Resistencia que posea no afecta al ataque que vaya a ejecutar mientras que la Fuerza sí. El estado entonces quedaría formado por:

$$< F, C, T_{NPC1}, T_{NPC2}, HP_{NPC1}, HP_{NPC2} >$$

siendo:

- **F :** Fuerza del personaje del agente humano
- **C :** Clase del personaje del agente humano
- **T_{NPC1} :** Tipo de enemigo 1
- **T_{NPC2} :** Tipo de enemigo 2
- **HP_{NPC1} :** Puntos de vida del enemigo 1
- **HP_{NPC2} :** Puntos de vida del enemigo 2

Al igual que en el estado del agente automático de aprendizaje, estos atributos van a tener determinados valores y algunos de ellos se tienen que agrupar para que la combinación de posibles valores no sea inmensa. Entre ellos, la clase de personaje y el tipo de enemigo se mantienen igual que antes, con un rango fijo de 3 y 6 valores respectivamente.

La Fuerza del personaje sigue el mismo número de divisiones que la Resistencia del personaje en los estados del agente automático de aprendizaje, pero esta vez, como el valor máximo es 20 en lugar de 15, sus valores en los grupos varían: el primer grupo empieza en 6 y acaba en 10, luego de 11 a 15 y, por último, 16 a 20. Los puntos de vida de los *NPC* coinciden con los establecidos en el estado del agente automático de aprendizaje.

Con el objetivo de explorar mayores posibilidades y de extender la cantidad de pruebas a realizar, se han utilizado un total de 2 agentes humanos diferentes. El primero efectúa acciones en función de dos jugadores que no tienen mucha experiencia con este tipo de juego y se podría decir que las acciones son casi aleatorias. Por otro lado, el segundo agente está creado desde cero a partir de reglas que determinan qué acción va a efectuar el mayor daño posible.

Para el **primer agente** se grabaron las acciones realizadas en varias partidas por dos jugadores inexpertos y se almacenaron en un archivo asociando la acción realizada con el estado del agente humano en el que se encontraban. Tras esas partidas, como no todos los estados se visitan, se completará utilizando acciones aleatorias. La compleción de todos los posibles estados no supone un gran porcentaje de ellos ya que en cada partida

se visitan una gran cantidad de estados y utilizando dos usuarios, las visitas a los estados se duplica en casi todas las ocasiones.

El **segundo agente**, a diferencia del primero, cubre todos los estados posibles desde el primer momento y no requiere de ningún proceso de completar la lista de estados. La construcción de los estados y acciones se elabora fuera de una partida y se rige por unas determinadas reglas. Las reglas se centran en aumentar el daño al máximo que pueda hacer el agente humano en cada una de las situaciones, por ello, entre otros factores, se analizan las relaciones clase-tipo de enemigo para ver si alguno es favorable para el agente. En teoría, este agente va a ejecutar las acciones que se consideran mejores para cada situación.

El propósito del primer agente es actuar como una simulación de un jugador muy inexperto mientras que el segundo agente tiene como objetivo simular un jugador más experimentado en este tipo de juegos y en particular en este juego y sus mecánicas.

Al final no se utiliza ningún tipo de clasificador al cubrir fácilmente el número de estados con ambas estrategias planteadas para los dos agentes humanos.

Ambos agentes se utilizarán en las pruebas realizadas y, tras obtener resultados, se compararán. La obtención de datos para los agentes, junto con otros datos, se explica más adelante en el apartado 5.2.

5.1.3. ENTRENAMIENTO

El entrenamiento del sistema es esencial para poder conseguir un agente funcional capaz de ejecutar acciones en función de su propio conocimiento y la información obtenida del entorno. Este proceso requiere la preparación de unas condiciones de entrenamiento que definan cómo va a efectuarse.

Como el entrenamiento está centrado en el combate, las condiciones tendrán que estar definidas en ese tipo de entorno. Por ello, se estructura un sistema de combate por turnos donde se enfrenten el agente humano previamente construido y el agente automático de aprendizaje. Con el objetivo de potenciar la exploración dentro del espacio de estados, cada partida empezará con valores aleatorios de todos los **atributos** definidos en el estado del agente automático. Esto llevará a que la duración de las partidas sea muy variada al poder empezar el enfrentamiento con *NPC* cuya vida está cercana a 0 y, por tanto, se podrá tener partidas de hasta 1 solo turno o partidas que lleguen a durar 6 turnos por cada agente.

Además de los atributos de los estados, se requiere establecer el valor de cada uno de los **parámetros del algoritmo de *Q-Learning***. Estos parámetros van a influir de forma directa a la forma de aprender del sistema y el enfoque a aplicar ante una disyuntiva que se le presente al agente. Estos parámetros son Alpha, Épsilon y Gamma.

El valor de **Alpha**, α, empezará con un valor muy cercano a 1 para potenciar la información externa (información proporcionada por el entorno) más que el conocimiento del agente. E irá disminuyendo hasta un valor cercano a 0 para dar más relevancia al

conocimiento previo de ese estado que a la recompensa obtenida tras visitarlo. Esto último se hace para centrarse en la explotación de un estado ya visitado.

Épsilon, ε, que determina si la acción elegida por el agente se ejecuta o se ejecuta la otra, empezará con un valor cercano a 1 para que se potencie la exploración al tener más probabilidades de ejecutar la acción contraria. E irá disminuyendo hasta alcanzar un valor muy cercano a 0 para que se efectúen las acciones que realmente se han escogido.

Por último, **Gamma**, γ, aunque hay casos en los que su valor puede ser dinámico en el entrenamiento, se elige tratarlo como un elemento fijo. En las pruebas, tomará distintos valores entre 0 y 1. Al tratarse de un valor fijo durante la misma sesión de aprendizaje, las pruebas pueden ir enfocadas en variar este valor para saber cuál proporciona mejores resultados.

El último elemento por fijar es el **número de sesiones** de entrenamiento que se le va a someter al agente. Por lógica, cuanto mayor sea ese número, mayor será lo que abarque en el espacio de estados. Este valor también definirá las pruebas que se ejecutarán con valores como: 50, 100, 500, 1000, 2000, 5000, 10000 y 15000.

5.1.4. PRUEBAS

Tras haber entrenado al agente, se debe estudiar los resultados dentro de una partida más cercana a la realidad. Como los atributos de los estados en el entrenamiento han obtenido valores aleatorios, no sería apropiado estudiar los resultados ya que, en muchas ocasiones, un bando estaría más desbalanceado que otro por empezar con un valor de vida muy inferior.

Por ende, para poder medir la calidad del entrenamiento se debe preparar otro entorno de enfrentamiento que vuelva a enfrentar al agente humano con el agente automático de aprendizaje. Pero esta vez, los atributos de vida de ambos bandos, humanos y *NPC*, empiezan en su valor más alto posible, manteniendo aleatorios el resto de los atributos (clases, tipos de enemigo, fuerzas y resistencias). Así representaría un combate desde el principio donde el enfrentamiento está balanceado y nivelado para ambos bandos.

El objetivo de este entorno es conseguir el porcentaje de victorias por parte del agente automático de aprendizaje para un número de partidas establecidas. Este entorno se debe utilizar para todas las sesiones de entrenamiento por separado.

Sin embargo, se puede decidir dar información parcial del juego a los *NPC*, omitiendo ciertos elementos existentes que puedan influir. Esta práctica se hace para dar una ventaja a los jugadores y para observar las respuestas del agente ante información que no corresponde con la realidad. En este caso particular, la información de las armas que poseen los jugadores no es transmitida a los agentes automáticos de aprendizaje.

Por esta razón, los entornos de enfrentamiento en este apartado van a dividirse en dos que se diferencian por los elementos utilizados en ellos. Para ser más específicos, en un entorno los personajes llevarán armas, generadas aleatorias por cada partida, mientras que en la otra no llevarán y los personajes efectúan 4 puntos de daño base. En el segundo caso, el agente automático de aprendizaje conoce el 100% de la información de los

jugadores exceptuando el número de pociones que no influye realmente en los resultados. Por otro lado, en el primer caso la información del arma no es conocida pero los agentes automáticos saben que el jugador hace 4 de daño, que puede que sea cierto, que sea menos o que sea más. Esto ocasiona que acciones que se tomen al final no resulten tan favorecedoras como inicialmente parecían.

Cada uno de estos dos entornos de enfrentamiento entre los dos agentes consistirá en 500 partidas. Para cada uno, se ejecutarán esas 500 partidas 5 veces y se hará una media del porcentaje de victorias para el agente automático de aprendizaje.

Además de utilizar estos dos entornos de enfrentamiento, se dispone de dos agentes humanos obtenidos de distintas formas que deben ser sometidos al mismo número de pruebas y con las mismas condiciones para poder compararlos. El primero, H1, es el agente que se ha construido con un simulador y el segundo, H2, es aquel construido mediante reglas.

Finalmente, se realiza una única prueba adicional con el máximo número de sesiones de entrenamiento para el caso en el que se haya obtenido el mejor resultado, pero aumentando ligeramente el valor de Gamma.

5.1.5. RESULTADOS

Al juntar todas las posibles pruebas establecidas en el apartado anterior y se almacenan los resultados de los porcentajes de victorias de los agentes automáticos de aprendizaje, se obtiene unas tablas como estas:

	PRUEBAS CON AGENTE HUMANO H1					
	gamma = 0.1		*gamma = 0.5*		*gamma = 0.9*	
Nº trainings	***% win (no wpn)***	***% win (wpn)***	***% win (no wpn)***	***% win (wpn)***	***% win (no wpn)***	***% win (wpn)***
50	38,2	37,5	38,5	37,8	37,0	36,4
100	38,7	38,3	39,0	38,6	38,9	37,5
500	40,1	39,3	40,7	40,1	39,3	38,0
1000	41,2	40,5	41,4	40,3	40,8	40,5
2000	45,7	44,2	47,3	45,9	46,4	45,8
5000	49,2	48,0	51,9	50,2	49,0	48,3
10000	53,4	51,7	54,5	53,8	53,6	52,0
15000	55,6	53,9	60,4	59,1	57,2	55,8

Tabla 63: Pruebas con agente humano H1

	PRUEBAS CON AGENTE HUMANO H2					
	gamma = 0.1		*gamma = 0.5*		*gamma=0.9*	
Nº trainings	***% win (no wpn)***	***% win (wpn)***	***% win (no wpn)***	***% win (wpn)***	***% win (no wpn)***	***% win (wpn)***
50	35,1	34,0	35,4	34,2	35,7	35,4
100	35,5	34,8	36,1	36,0	36,2	35,8
500	37,2	35,4	37,6	36,9	37,5	36,3
1000	39,7	37,2	40,0	38,6	40,1	38,2
2000	41,0	40,7	44,1	42,8	42,0	41,6
5000	47,8	46,3	47,8	47,0	47,4	46,1
10000	50,1	49,6	51,5	50,3	50,4	49,0
15000	51,5	51,0	55,2	53,9	52,6	50,8

Tabla 65: Pruebas con agente humano H2

*wpn: *Weapon*, 'arma' en inglés

*win: 'victoria' en inglés

'*No wpn*' implica que la información que tiene el agente es total, mientras que, con '*wpn'* el agente tiene información parcial.

Para cada una de las pruebas se puede obtener una gráfica que represente el avance de victorias a medida que aumenta el número de sesiones de entrenamiento. Cada gráfica está fijada por un valor de Gamma que no varía por lo que habrá dos evoluciones en cada gráfica: una en la que los personajes tengan armas y otra en la que no, es decir, una en la que el agente automático no tenga toda la información del juego y otra en la que sí, respectivamente. Cada gráfica se compone por el porcentaje de victoria en el eje vertical y número de sesiones en el eje horizontal.

Con el agente humano H1 se obtienen las siguientes gráficas dependiendo del valor de Gamma en las figuras 16, 17 y 18.

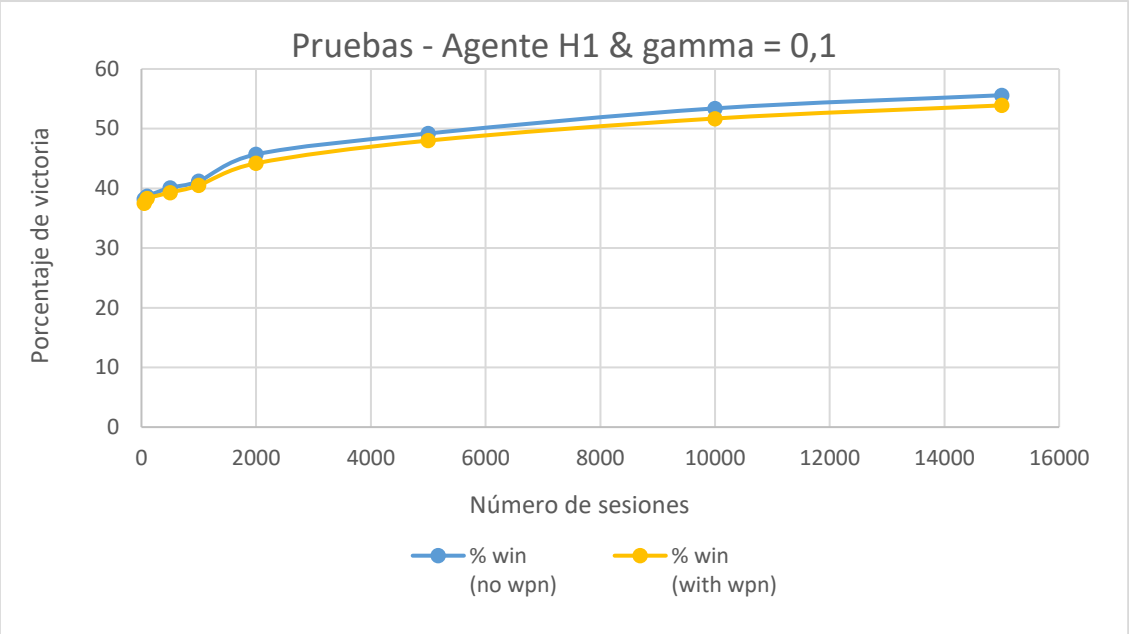

Figura 16: Pruebas con agente humano H1 y gamma 0,1

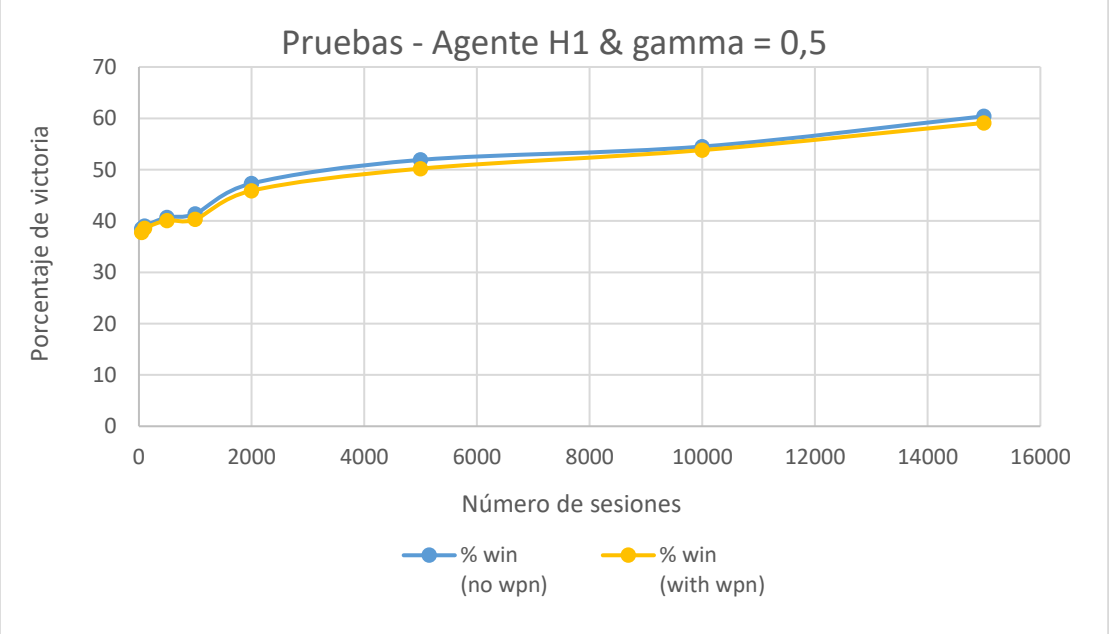

Figura 17: Pruebas con agente humano H1 y gamma 0,5

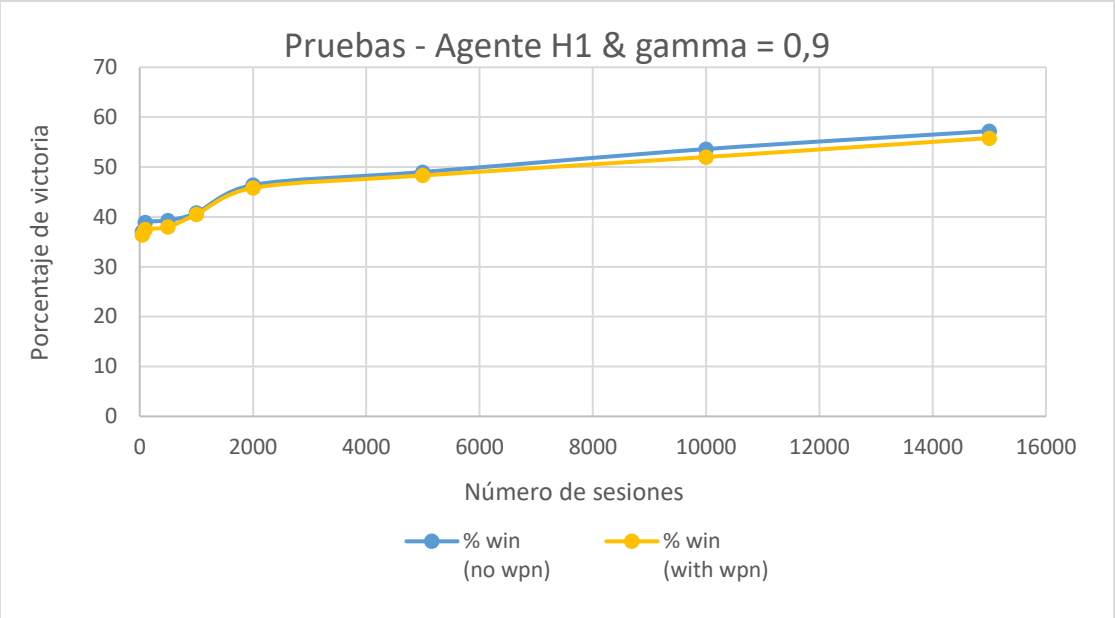

Figura 18: Pruebas con agente humano H1 y gamma 0,9

Mientras que con el agente humano H2, las gráficas obtenidas según el valor de Gamma son las siguientes mostradas en las figuras 19, 20 y 21.

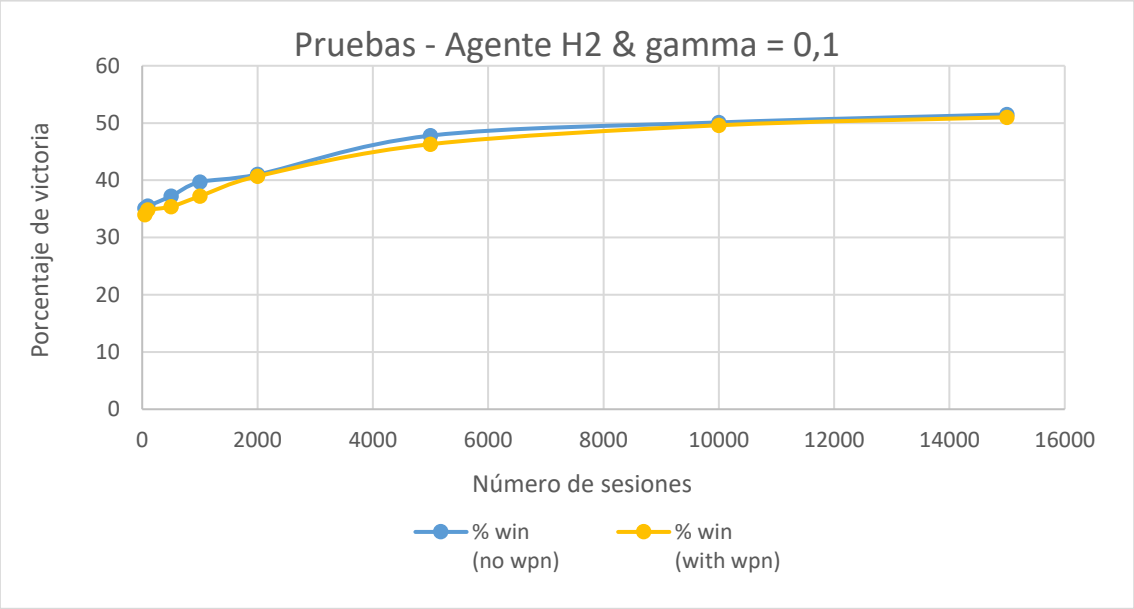

Figura 19: Pruebas con agente humano H2 y gamma 0,1

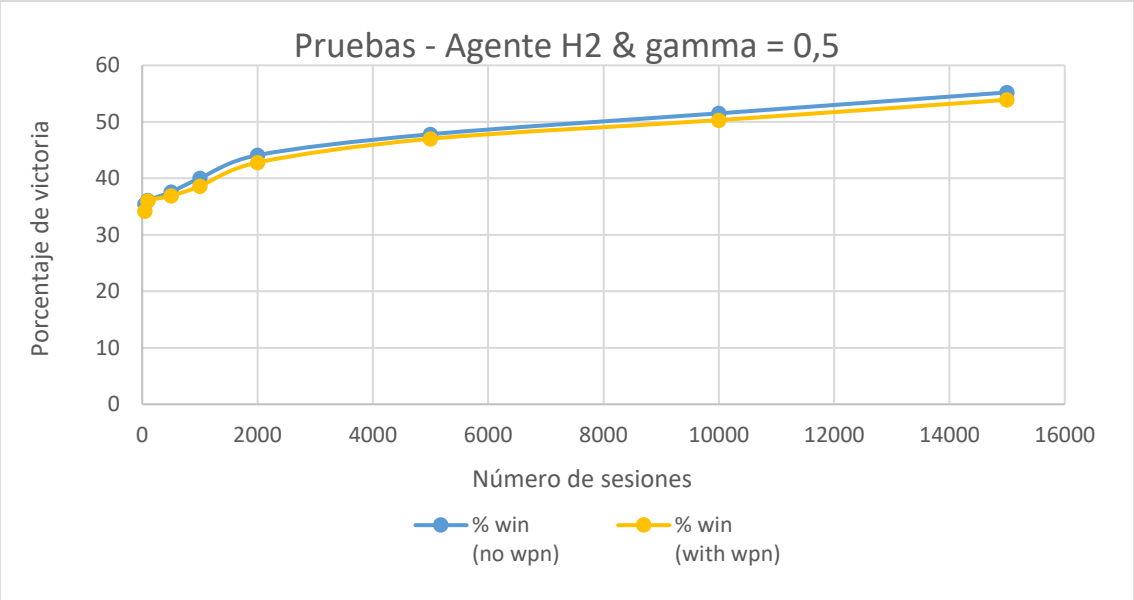

Figura 20: Pruebas con agente humano H2 y gamma 0,5

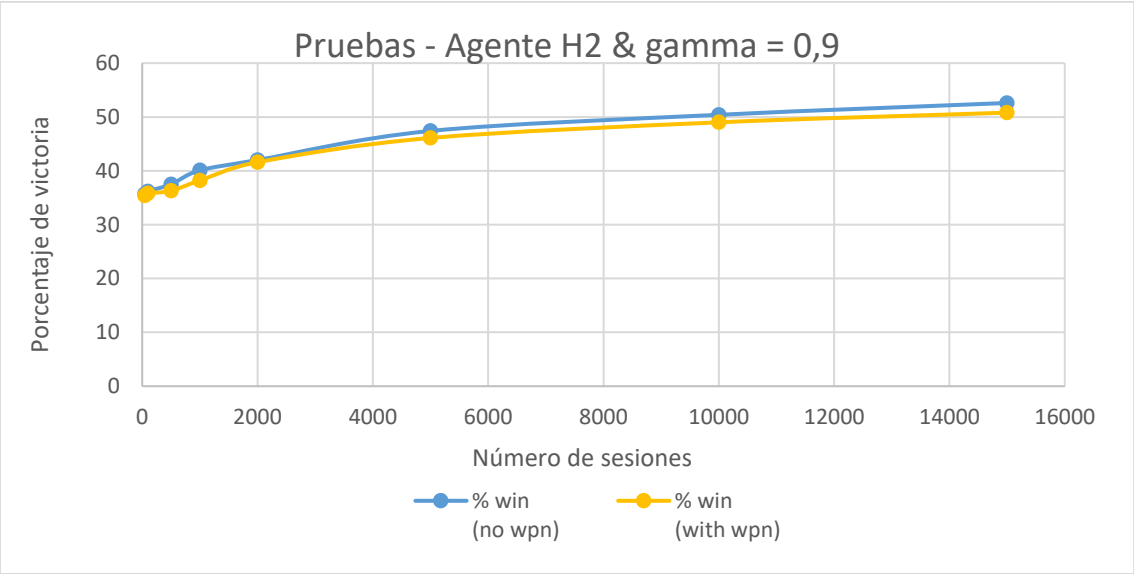

Figura 21:Pruebas con agente humano H2 y gamma 0,9

Tal y como se puede apreciar en las todas las gráficas, las victorias del agente automático de aprendizaje van aumentando a medida que lo hace el número de sesiones de entrenamiento. Como es lógico, cuantas más sesiones de entrenamiento se utilicen, más estados podrá explorar el agente y además obtendrá mejor precisión en la explotación de ciertos estados.

Las seis gráficas expuestas presentan un avance muy similar diferenciándose ligeramente en los valores máximos alcanzados en cada una de las sesiones. Empezando, generalmente, con un tercio de las victorias y acabando con, en muchos casos, un valor muy cercano al doble del porcentaje inicial.

5.1.5.1. CONOCIMIENTO DEL ENTORNO

En este apartado es necesario hacer alusión a la discusión previamente planteada del uso de dos situaciones diferentes en cuanto a la información que conocía el agente automático para el estudio de la calidad del entrenamiento. Sin realizar ningún tipo de prueba, se expuso de forma teórica que cuando el agente automático de aprendizaje se encuentra en un entorno del cual conoce toda la información, este agente es capaz de efectuar las acciones de forma más controlada y con mejor precisión obteniendo, entonces, unos resultados más favorables. Mientras que, también de forma teórica, si el agente automático se encuentra en un entorno del cual conoce una información parcial siendo la parte desconocida uno o varios elementos cruciales para la toma de elecciones, el agente también es capaz de elegir qué acción efectuar, pero al desconocer ciertos aspectos estas acciones pueden repercutir en un efecto negativo sin haberlo previsto por el agente. Por tanto, el primer caso, en teoría, debería otorgar unos resultados más favorables que el segundo caso.

Cuando se lleva a la práctica se puede apreciar que, con información total, el agente obtiene mejores resultados que cuando se tiene información parcial del entorno. Sin embargo, la diferencia de los resultados de victorias entre estas dos situaciones es muy pequeña llegando a tener solo 1% de separación en las sesiones más altas. Ese valor solo supone, en un total de 500 partidas que componen el enfrentamiento cerrado, 5 partidas menos ganadas.

Aun así, es lógico que la separación entre estas dos situaciones no sea tan grande por una cuestión a plantear, ¿a quién favorece la información que se desconoce? Partiendo que lo único que desconoce el agente automático es el arma que tiene el personaje, se debe observar qué efecto produce este desconocimiento. Al entrenar al agente automático de aprendizaje se le establece que los personajes van a efectuar un daño base de 4 puntos sin aplicar ningún tipo de bonificación por clase ni por Fuerza. Este valor, ahora incluyendo las bonificaciones de daño, es el que los agentes saben que el humano va a realizar, pero solo si lo establecemos como si fuese la información total que sabe dicho agente.

Ahora, cuando se añade la información adicional de que el personaje no siempre va a hacer 4 puntos de daño base más las bonificaciones con un elemento del cual el agente no tiene información, la situación cambia. Pero el detalle que falta por mencionar en esta información es qué rango de valores pueden tener las armas. Aunque este detalle se ha mencionado en el apartado 4.1.3., las armas pueden tener un daño base entre 2 y 6 incluidos ambos. Si lo comparamos con el daño base que se efectúa en el entrenamiento, 4 puntos, este rango favorece en la mitad de los casos a los personajes y en la otra mitad a los agentes automáticos. Al efectuar el personaje un daño inferior al esperado, el agente automático, aunque no se espere un resultado como ese, sigue ejecutando las acciones que crea conveniente resultando en más victorias, ya que el personaje va a hacer menos daño y los *NPC* morirán más tarde, o directamente los personajes son derrotados antes de que consigan matar a los *NPC*.

Además del daño base, cada arma proporciona una ventaja adicional al jugador como ya se explica en el apartado 4.1.3. siendo ventajas de daño a ciertos tipos de enemigo o un aumento numérico en uno de los atributos del jugador. Por tanto, no sólo se aumenta el

daño base, sino que las bonificaciones de daño también aumentan. Aunque esta cualidad es menos tratada ya que al tener efecto con un tipo de enemigo en especial, se dará en menos ocasiones y se explotará menos.

Por ende, al tratarse de armas que oscilan su daño base dos puntos por debajo o por encima del daño base establecido en el proceso de aprendizaje, los resultados son menores al desconocer esa información, pero como la diferencia de daño no es tan significativa los resultados se asemejan. Si entramos en profundidad, la separación de estas situaciones no es tan grande ya que, en muchas ocasiones, que un personaje efectúe 4 ó 6 puntos de daño lleva a una misma situación, es decir, lleva a un mismo estado del agente automático. Por ejemplo, si un *NPC* tiene 9 puntos de vida y el personaje realiza 4 puntos de daño, esto lleva a un estado del agente automático que sería el mismo que si el daño realizado fuese de 5 o de 6. Y si el *NPC* tiene 2 puntos de vida, independientemente del daño base, el estado final va a ser el mismo: el personaje derrota al *NPC*.

5.1.5.2. AGENTES HUMANOS

Por otro lado, es necesario el análisis de los resultados obtenidos de forma conjunta en cada uno de los escenarios donde se utilizan distintos agentes humanos para el enfrentamiento del entrenamiento. Analizando ambos agentes de forma teórica, el agente humano H2, construido mediante el uso de reglas, debería vencer en un número elevado de ocasiones al analizar cuál será la acción que efectúe mayor daño en todas las ocasiones gracias al sistema de reglas establecido. Por otro lado, el agente humano H1, al estar simulando dos jugadores que no tienen mucha experiencia en este juego, no abarca el 100% de los estados posibles y muchas acciones extraídas a partir de esta simulación puede que corresponda con acciones tomadas de forma aleatoria. Por ello, en teoría, este agente humano debería ganar menos partidas que el otro agente humano.

Al llevarlo a la práctica, tras ambos entrenamientos y enfrentamientos en las partidas de prueba, la diferencia del porcentaje de victorias para los agentes automáticos de aprendizaje entre el entrenamiento con el agente humano H1 y el agente humano H2 es mínima, aunque el agente humano H2 gane más partidas que el agente humano H1. Se mantiene la deducción obtenida de forma teórica pero falta analizar por qué la diferencia es tan pequeña. Lo primero de todo, se está tratando un problema de decisión en donde las alternativas del agente automático no son muy numerosas. Esto implica que los distintos resultados obtenidos desde un mismo estado aplicando las distintas acciones disponibles se reduzcan a un número bajo. Para ser más precisos, en un enfrentamiento de 2 contra 2, a partir de las acciones efectuadas por ambos agentes automáticos, se presentan 4 posibles situaciones:

- *NPC* 1 ataca a Personaje 1 y *NPC* 2 ataca a Personaje 1
- *NPC* 1 ataca a Personaje 1 y *NPC* 2 ataca a Personaje 2
- *NPC* 1 ataca a Personaje 2 y *NPC* 2 ataca a Personaje 1
- *NPC* 1 ataca a Personaje 2 y *NPC* 2 ataca a Personaje 2

Pero también se puede dar el caso de que uno de los *NPC* esté derrotado y no pueda ejecutar ninguna acción dando otras 4 situaciones:

- *NPC* 1 ataca a Personaje 1 y *NPC* 2 esté derrotado
- *NPC* 1 ataca a Personaje 2 y *NPC* 2 esté derrotado
- *NPC* 2 ataca a Personaje 1 y *NPC* 1 esté derrotado
- *NPC* 2 ataca a Personaje 2 y *NPC* 1 esté derrotado

Al tener solo 4 diferentes resultados con dos acciones disponibles (atacar a personaje 1 o a personaje 2) para cada agente automático, realizar una acción diferente de la más apropiada puede que lleve también a un resultado favorable, pero en un número de pasos mayor. Toda acción efectuada a lo largo del enfrentamiento influye en el resultado final, pero eso no significa que por realizar una acción que no es la más provechosa en esa situación, el agente automático vaya a perder seguro la partida. Aunque, finalmente, en casos donde se han elaborado más sesiones de entrenamiento, la diferencia entre ambas es de hasta un 6% que supone 30 partidas ganadas más.

Además de este aspecto, es muy importante también aludir al entrenamiento de los dos tipos en función del agente humano utilizado. Aunque ambos compartan las condiciones de entrenamiento, tanto el procedimiento y el número de sesiones, los datos de entrenamiento utilizados son distintos en cada uno de ellos por el hecho de que los datos del problema se generan de forma aleatoria al empezar la partida. Se podría plantear la creación de un conjunto de estados fijos para los datos de entrenamiento con el fin de que sean iguales en ambos casos, pero entonces no se seguiría el proceso de entrenamiento por refuerzo donde en cada sesión se empieza por un estado aleatorio para potenciar tanto la exploración como la explotación del problema.

5.1.5.3. VALOR DE GAMMA

Finalmente, el último aspecto que diferencia a cada una de las pruebas realizadas y que se debe analizar es el valor asociado a **Gamma**. Este valor, que determina la importancia de las recompensas de forma directamente proporcional a su valor, tendrá 3 posibles valores relevantes para una evaluación lo más extensa posible de la batería de pruebas. Estos valores serán uno cercano a su límite inferior (0,1), un valor intermedio (0,5) y uno cercano a su límite superior (0,9). Los resultados obtenidos en ambas pruebas con distintos agentes humanos experimentan una evolución similar en sus valores en función del valor de Gamma por lo que este análisis cubre ambas pruebas.

Como podemos ver, el valor máximo alcanzado en el porcentaje de victorias se alcanza cuando Gamma obtiene el valor intermedio de 0,5. Se podría pensar que cuanto mayor sea el valor de Gamma, los resultados deberían mejorar de forma linealmente proporcional pero no se cumple en cualquier caso y aquí se demuestra. Como delimita cómo de importantes son las recompensas futuras que recibirá el agente, si utilizamos un valor muy cercano a 1, su límite superior, todas esas recompensas futuras se valorarán de la misma forma por el agente. Esto significa que si, por ejemplo, en 10 acciones el agente realiza algo beneficioso se le da tanto valor como si lo hubiese realizado en 1 acción y esto ocasiona un impacto en los resultados. En muchos casos, como es este, el aprendizaje no trabaja tan bien con valores tan altos de Gamma.

En el caso opuesto, donde se utiliza un valor de Gamma muy cercano a 0, el agente solo tendrá en cuenta los refuerzos inmediatos, que puede funcionar para funciones de refuerzo muy específicas, pero no para la establecida en este problema.

Finalmente, con respecto al caso de la **prueba única adicional**, al tratarse del mejor resultado en el caso de Gamma con 0,5, se aumenta a 0,75. Tras realizar las mismas verificaciones que con el resto de las pruebas, obtenemos un porcentaje de victorias del agente automático de 56,9 % cuando se utilizan armas y 58,6 % cuando no se utilizan.

Por lo que, para 15,000 sesiones y el agente humano H1, se puede apreciar una evolución de los porcentajes de victoria en función de Gamma que progresa en la figura 22.

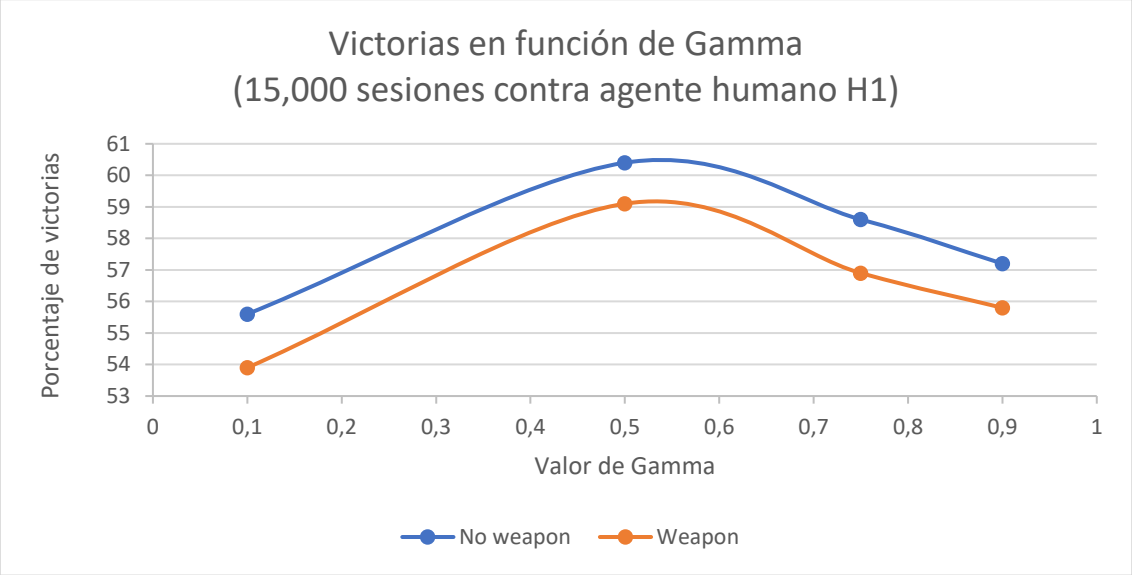

Figura 22: Comparación de victorias en función de Gamma

Para muchos otros casos de aprendizaje por refuerzo a medida que se avanza por el valor de Gamma, los resultados son más favorecedores guardando una relación directamente proporcional entre ambos aspectos. Sin embargo, estos valores dependen del enfoque del problema y no siempre se encuentran comportamientos de este tipo. Un buen ejemplo es este proyecto, que alcanza unos resultados favorables cuando obtiene un valor alto, pero sin llegar a acercarse al límite superior.

5.2. FUENTES DE DATOS

En este proyecto se requieren determinados datos que se utilizarán para funcionalidades particulares del sistema. Estas funcionalidades cubren en su totalidad el aprendizaje del agente automático e influyen en los resultados obtenidos tanto en el entrenamiento como en los análisis posteriores.

A pesar de haber explicado estos conceptos previamente, se entrará en detalle en la obtención de todos los conjuntos de datos en este apartado.

El primer conjunto de datos relevantes para el aprendizaje está dedicado a la simulación de los agentes humanos que actuarán de forma automática tanto en el entrenamiento y en el posterior entorno de evaluación.

Como se ha mencionado previamente, se han separado los conjuntos de estados de los agentes humanos en dos: uno creado para simular a un jugador experto y otro para simular un jugador inexperto. Esta separación se hace clara y bien marcada para demostrar que el comportamiento del contrincante del agente automático influye de forma directa en los resultados de una partida. El segundo agente no se obtiene de ninguna fuente, sino que se genera a partir de reglas que establecen en cada momento cuál será el daño máximo que puede efectuar ese personaje a cualquiera de los contrincantes *NPC* a los que se enfrenta. Pero el primer agente sí requiere un proceso de obtención.

Para obtener los datos que simulen jugadores inexpertos se requiere de al menos un jugador que sea inexperto para jugar y captar sus acciones en cada momento. La solución ha sido la colaboración de dos usuarios voluntarios cuyo conocimiento en juegos de este estilo es bastante bajo. Para poder simular las acciones a tomar, se les ha preparado un entorno exclusivo de combate donde, antes de empezar, se les dejaba crear sus personajes con indicaciones para facilitar la mayor extensión de visita de estados posible. Habiendo creado los personajes, se han jugado 20 partidas en el entorno de combate y para cada turno se han guardado el estado y la acción efectuada. Cada partida empieza con ciertos valores aleatorios como, por ejemplo, los tipos de enemigos generados. Tras las 20 partidas, se tendrá un archivo con bastantes estados y acciones correspondientes obtenidas de los usuarios, pero al no llegar a la totalidad de estados posible, se ejecutará un programa externo que no requiera de los usuarios para completar el resto de los estados sin visitar y estableciendo acciones aleatorias. Esto se hace, fundamentalmente, para hacer hincapié en que este agente humano sea lo más fácil posible para enfrentar al agente automático contra él. Los usuarios presentes en la obtención de estos datos siguen estas características:

- ***ID***: Valor alfanumérico de los voluntarios. Compuesto por '*voluntX*' siendo X el índice numérico empezando por 1.
- ***Edad***: Representa la edad de los voluntarios
- ***Sexo***: Con valores válidos de Mujer y Hombre
- ***Experiencia con juegos***: Recoge si cada voluntario ha jugado a algún juego de este tipo previamente y cómo de bien se manejan con ellos. Tendrá valores de ninguna, baja, media y alta.

ID	*Edad*	*Sexo*	*Experiencia con juegos*
volunt1	56	Hombre	Ninguna
volunt2	7	Mujer	Baja

Tabla 66: Voluntarios para agente humano H1

El segundo conjunto encuadra los distintos estados que visita el agente automático durante el entrenamiento y el entorno de evaluación. Como se ha mencionado en momentos anteriores, este conjunto de datos realmente no se obtiene de ningún lado ya que se genera de forma aleatoria con el objetivo de alcanzar el máximo número de estados

potenciando la exploración, pero, a su vez, poder revisitar un mismo estado para potenciar la explotación.

5.3. EVALUACIÓN DE PRUEBAS DEL SISTEMA

Antes de realizar una evaluación utilizando usuarios de prueba, se requiere llevar a cabo las pruebas expuestas en el apartado de Análisis. Para su ejecución, se harán modificaciones en el código para alcanzar la situación determinada que define a cada una de las pruebas. Lo único que se debe destacar en este apartado es si la prueba ha sido satisfactoria, es decir, si se ha obtenido el resultado esperado establecido en Análisis.

PruebaID	*Satisfactoria*
PR-01	Sí
PR-02	Sí
PR-03	Sí
PR-04	Sí
PR-05	Sí
PR-06	Sí
PR-07	Sí
PR-08	Sí
PR-09	Sí
PR-10	Sí
PR-11	Sí
PR-12	Sí
PR-13	Sí
PR-14	Sí

Tabla 67: Resultados pruebas del sistema

Todas las pruebas efectuadas nos daban el resultado deseado establecido en el apartado de Análisis y seguían el procedimiento esperado.

5.4. USUARIOS DE PRUEBA

Después de haber implementado tanto el código de juego como la estructuración del aprendizaje del agente automático, es necesario realizar una evaluación objetiva de los resultados. No sólo se debe analizar la calidad y dificultad del agente automático que se enfrenta a los jugadores, sino que también se tiene que tener en consideración la jugabilidad y la claridad dentro del avance de una partida. Por ello, la prueba a ejecutar es jugar una partida hasta que se gane o pierda.

Construir un juego que sea fácilmente entendible para todo tipo de jugadores es una tarea ardua que requiere un gran número de revisiones y modificaciones en determinadas partes y procesos para conseguir la mayor claridad a la hora de jugar y, fundamentalmente, para que resulte divertido.

Por ello, se ha recurrido a un número de usuarios externos al proyecto que no conozcan nada del juego en sí para que experimenten objetivamente el avance por una partida normal. Las condiciones iniciales a las que se somete a los usuarios son comunes para todos, aunque el avance por distintas partes del juego va a depender completamente de los jugadores involucrados.

Cada partida se jugará con dos jugadores diferentes que realicen acciones en conjunto y de forma particular. El objetivo de esta evaluación no es comprobar cómo de bien juegan estos usuarios sino controlar que en cada momento se están enterando de lo que está pasando, que el juego les resulte al menos entretenido y de revisar la facilidad de las actividades que realizan.

Con el propósito de alcanzar el máximo número de arquetipos de usuario, se han escogido usuarios característicos que se diferencian generalmente por sus experiencias y habilidades con juegos de este estilo. Esto facilitará que, en el estudio de calidad del juego, se tengan opiniones de todo tipo de jugadores para realizar modificaciones en él y que pueda ser utilizado por el mayor número de personas.

Cada usuario de prueba estará caracterizado por los siguientes rasgos:

- ***ID***: Valor alfanumérico de los voluntarios. Compuesto por '*testUserX*' siendo X el índice numérico empezando por 1.
- ***Edad***: : Representa la edad de los usuarios de prueba
- ***Sexo***: Con valores válidos de Mujer y Hombre
- ***Experiencia con juegos***: Recoge si cada usuario ha jugado a algún juego de este tipo previamente y cómo de bien se manejan con ellos. Obtiene valores de ninguna, baja, media y alta.
- ***Clase elegida***: Entre las marcadas en la sección de elección de personajes dentro de la partida se encuentran Oso, Grifo y Lobo.

ID	*Edad*	*Sexo*	*Experiencia con juegos*	*Clase elegida*
testUser1	22	Hombre	Alta	Grifo
testUser2	58	Mujer	Ninguna	Grifo
testUser3	56	Hombre	Ninguna	Oso
testUser4	21	Mujer	Media	Lobo
testUser5	7	Mujer	Baja	Lobo
testUser6	15	Hombre	Media	Grifo

Tabla 68: Usuarios de pruebas

El rango de edades es lo más extenso posible porque, por norma general, este valor está relacionado con el apartado de *Experiencia con juegos*. Por lo general, la gente joven ha tenido un acceso más fácil a nuevas tecnologías y a videojuegos mientras que hace unos años esto no era tan común.

Es importante que antes de empezar la partida, se les debe dar un breve tutorial de cómo pueden realizar ciertas acciones y las mecánicas del juego. La cantidad de información dada no puede ser muy escueta ya que hay ciertos elementos que requieren de un

conocimiento previo, pero tampoco puede ser mucha información porque podría afectar a los resultados obtenidos en la evaluación de los usuarios de prueba. Los usuarios se agruparon en 3 partidas de jugadores.

La primera partida se jugó con los usuarios *testUser1* y *testUser4*. El progreso a través de la partida fue bastante fluido y fugaz, ya que ambos habían tenido experiencias no solo con videojuegos sino con los conceptos establecidos en las mecánicas. Conceptos como atributos, clases y tipos de enemigo no necesitaron explicación más allá de la inicial.

Por otro lado, en la segunda partida, se jugó con los usuarios *testUser2* y *testUser3* que no tenían ninguna experiencia previa ni con videojuegos ni con aspectos involucrados en ellos. Al igual que el resto de las pruebas de las partidas, la diferencia de habilidad y experiencia en los juegos entre los dos participantes no era muy significativa ya que las acciones de aquel más experimentado podrían influir en las del otro y sesgar el estudio. Aunque se avanzó de forma más calmada a lo largo de la partida, el resultado final fue satisfactorio y no hubo necesidad de explicaciones más detalladas.

El último grupo consiste en los usuarios *testUser5* y *testUser6* que, aunque no tenían tanto experiencia como el primer grupo, estaban familiarizados con la terminología y las mecánicas. Esto facilitó el avance por la partida de forma fluida y no se requiso explicaciones adicionales.

Como se puede ver, se ha intentado juntar en una misma partida a usuarios cuya experiencia fuese similar para que puedan actuar de una forma similar y, sobre todo, para poder establecer una colaboración equilibrada. Al tener elección en conjunto en muchas situaciones de la partida, se requiere que los jugadores discutan las opciones y elijan cuál es la más apropiada para ellos; y si ambos poseen un mismo conocimiento, las decisiones se vuelven más interesantes sin que ninguno influya demasiado sobre el otro.

5.5. FORMULARIO DE EVALUACIÓN

Después de las pruebas realizadas a los usuarios, se les debe consultar ciertos aspectos sobre su experiencia a lo largo de la partida. Para obtener ese *feedback* se ha recurrido a un formulario con diferentes preguntas sobre el juego y la partida que han jugado.

Para realizar estos formularios se ha recurrido a *Google Forms* (formularios de Google) que permiten a los usuarios responder a las preguntas establecidas y posteriormente obtener gráficos y comparaciones para el estudio de la evaluación.

Las preguntas planteadas en el formulario serán de varios tipos, incluyendo opciones a marcar y preguntas de escribir. Las preguntas expuestas en el formulario estarán divididas en dos secciones focalizadas en aspectos diferentes. La primera sección está centrada en la experiencia dentro de la partida:

- **Con respecto al avance por la partida, ¿te ha resultado fácil entenderlo?** La respuesta puede ser “Sí” o “No”
- **Con respecto a las mecánicas del juego (clases, tipos de enemigo, combate por turnos…), ¿te han resultado fácil entenderlas?** La respuesta puede ser “Sí” o “No”

- **¿Ha habido algún momento de la partida donde no sabías qué tenías que hacer? En caso afirmativo marcar dónde**. La respuesta puede ser "Sí" o "No", además se tiene que escribir, en caso afirmativo, en qué momento se ha sentido perdido/a.
- **¿Te ha resultado difícil el enfrentamiento contra los enemigos en el combate? Marque el nivel de dificultad siendo 1 muy fácil y 5 difícil.** La respuesta obtiene 5 posibles valores
- **¿Te ha parecido entretenido?** La respuesta puede ser "Sí" o "No"

Mientras que la segunda sección está centrada en los juegos en general y experiencias futuras ajenas a la partida que hayan jugado. Esta sección está pensada para conocer qué impacto tienen los videojuegos en distintos arquetipos de gente que actualmente viven en un mundo donde los videojuegos están tan al alcance de todos:

- **¿Consideras los videojuegos o juegos en general un método de entretenimiento apropiado?** La respuesta puede ser "Sí" o "No"
- **En caso de no tener mucha interacción con videojuegos, ¿qué te limita a jugarlos?** El usuario debe escribir una respuesta

Después de haber completado el formulario de todos los jugadores implicados en las partidas de pruebas, se puede elaborar una comparación y un estudio gracias a las respuestas y los resultados obtenidos.

Este estudio va a facilitar una serie de cambios en el juego con el objetivo de mejorar el juego en un aspecto general. Claramente, los cambios van dirigidos de forma particular a rasgos del juego que los usuarios han considerado que necesitan cierto repaso o rasgos que requieren un cambio radical. Al igual que una empresa se vuelca en sus clientes, un juego debe volcarse en sus clientes, es decir, sus jugadores porque son ellos los que van a experimentarlo. Por ello, las preguntas establecidas en el formulario abarcan cada uno de los aspectos más relevantes en una partida para conocer la opinión de los jugadores sobre dichos aspectos.

Tras los resultados se pueden obtener unas gráficas que representan las opiniones de los usuarios de prueba con respecto a cada una de las preguntas.

En el primer apartado, en las figuras 23, 24 y 25, las preguntas que tratan de la comprensión del juego tanto del avance como las mecánicas del juego han obtenido unos resultados muy similares sin destacar una respuesta fuera de lo común.

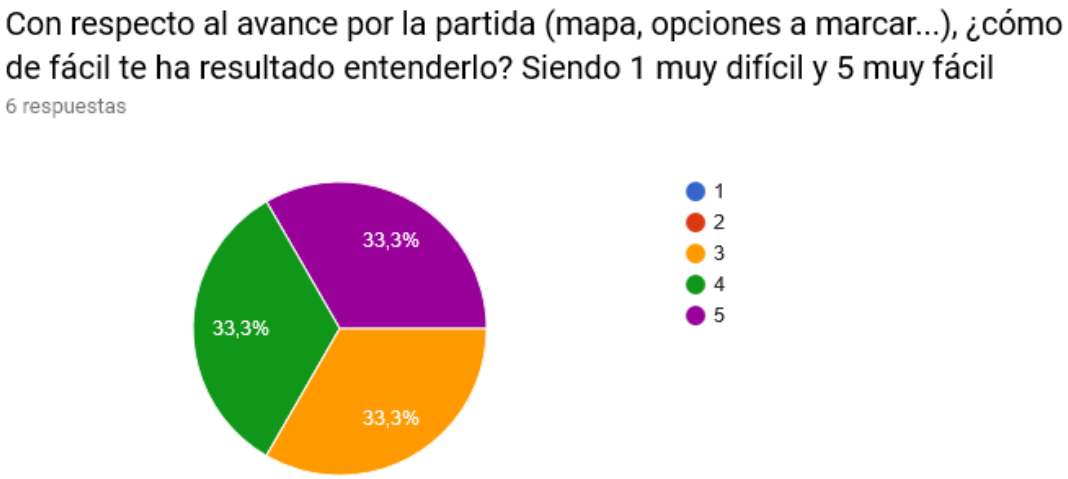

Figura 23: Gráfico de la primera pregunta del formulario de usuarios de prueba

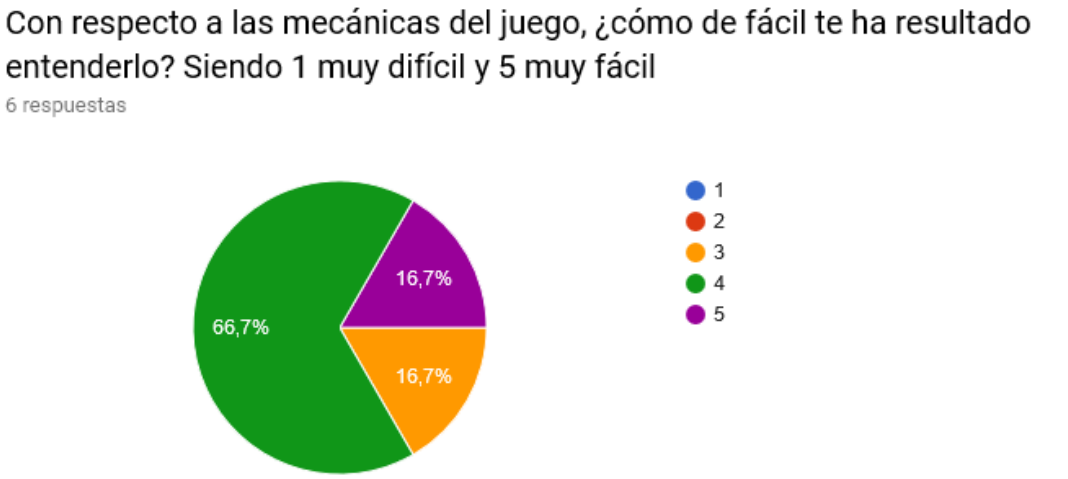

Figura 24: Gráfico de la segunda pregunta del formulario de usuarios de prueba

Con estos resultados podemos marcar que todos los arquetipos de usuarios que han participado han comprendido cómo funciona tanto el progreso a través de una partida como el método empleado para el enfrentamiento contra la máquina y sus mecánicas.

Además, el 100% de los resultados de la tercera pregunta afirman no haberse sentido perdidos en ningún momento. Esto significa que, para cada momento de la partida, las opciones a realizar están claras permitiendo al usuario mantener un avance continuado sin tener que pararse a buscar qué se debe hacer.

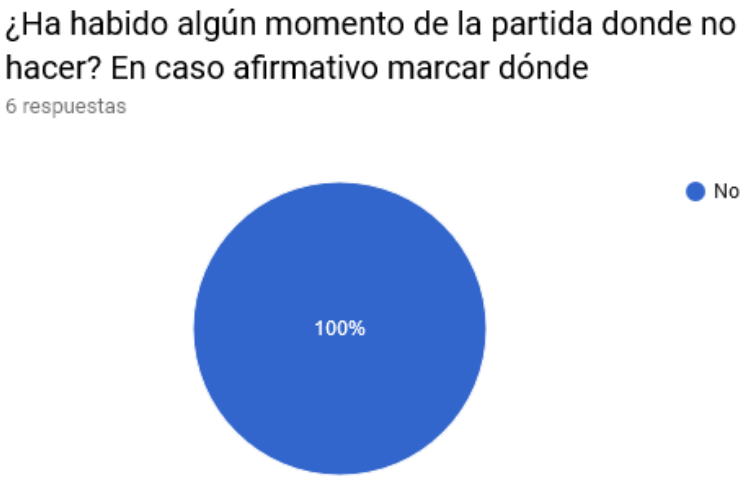

Figura 25: Gráfico de la tercera pregunta del formulario de usuarios de prueba

La siguiente pregunta está centrada en ver si los combates contra la IA han sido fáciles o les ha costado enfrentarse a ellos. Como ya se ha mencionado previamente, el agente automático no está centrado en vencer el máximo número de veces sino de hacer que los jugadores encuentren el enfrentamiento desafiante, pero a su vez se encuentren cómodos y no les resulte exageradamente difícil. Por lo general, a los usuarios les ha resultado medianamente difícil sin llegar a niveles muy altos como se ve en la figura 26.

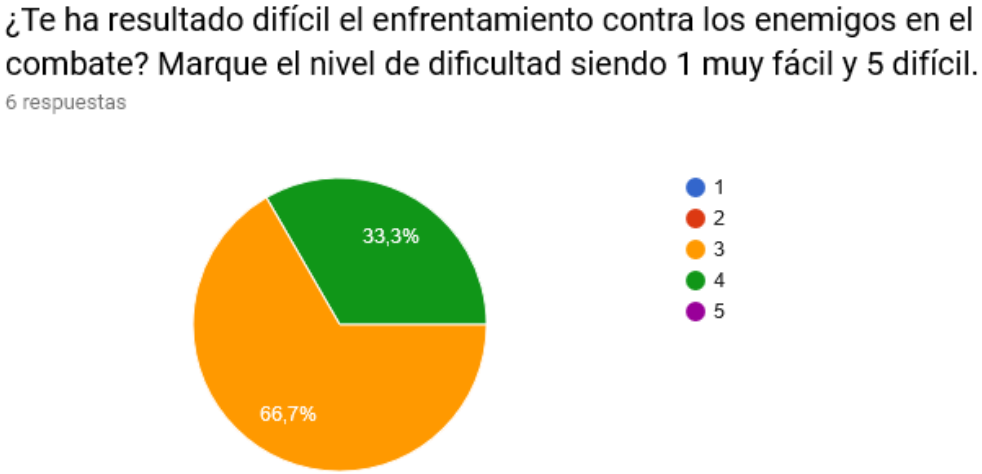

Figura 26: Gráfico de la cuarta pregunta del formulario de usuarios de prueba

Pasando a la segunda sección del formulario, ningún usuario pensó que los videojuegos son un método de entretenimiento poco apropiado tal y como se ve en la figura 27. Los resultados obtenidos son sorprendentes ya que casi la mitad de los usuarios no tienen experiencia con los videojuegos, pero aun así piensan que éstos son un buen método de entretenimiento.

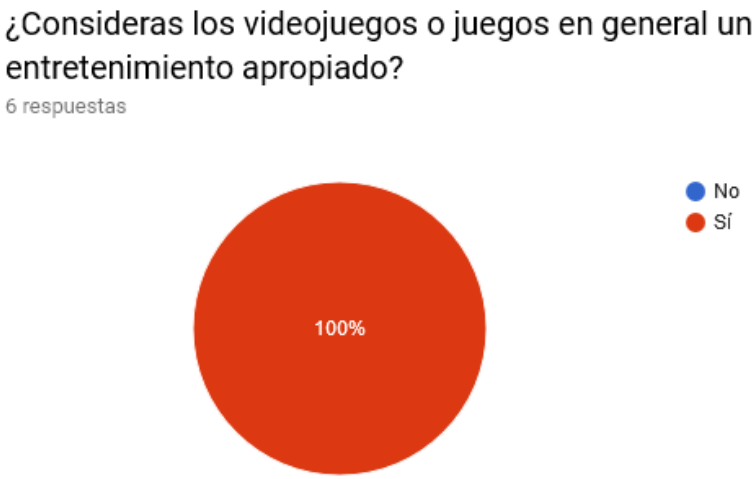

Figura 27: Gráfico de la pregunta adicional del formulario de usuarios de prueba

Los últimos resultados producen mayor asombro cuando estos usuarios son preguntados por las razones que los limitan a no jugar videojuegos actualmente. En la mayoría de los casos, se trata de personas que superaron la adolescencia y las etapas de juventud antes del crecimiento de los videojuegos. Por ello, una respuesta recibida trataba a los videojuegos de aburridos y otra marcaba que el manejo de los videojuegos puede resultar algo difícil para alguien que no ha jugado casi nada.

6. LEGAL

Esta sección está dedicada en su totalidad al estudio de los aspectos legales por los cuales se rige este proyecto para poder estar en regla con la legislación actual.

El primer aspecto legal existente en este proyecto es la Ley de Protección de Datos donde en el ámbito nacional existe la *Ley Orgánica de Protección de Datos* (LOPD). Según el Artículo 18.4 de la Constitución Española, esta ley limita el uso de la informática para garantizar el honor y la intimidad personal y familiar de los ciudadanos y el pleno ejercicio de sus derechos. Junto con la sentencia del Tribunal Constitucional se persigue garantizar a la persona un poder de control sobre sus datos personales, su uso y destino [39]. Sin embargo, esta ley quedó superada en 2016 por el nuevo *Reglamento General Europeo de Protección de Datos* (RGPD) que tiene una mayor área de efecto que la LOPD y las sanciones son mucho más estrictas. Este reglamento regula el tratamiento que realizan empresas y organizaciones de datos personales que guardan relación con personas de la Unión Europea. Además, delimita que los datos personales serán tratados de forma legal e ilícita en relación con el interesado y tendrán un propósito legítimo. Estos datos serán tratados de tal manera que garanticen la seguridad adecuada a datos personales incluida la protección contra actos ilícitos, pérdida o destrucción. El tratamiento de dichos datos será ilícito si protege los intereses vitales del interesado, entre otras condiciones. [40]

A pesar de tener una información mínima de los usuarios y jugadores del sistema de este proyecto, la presencia de este reglamento en el proyecto es necesaria ya que cada usuario que utilice el sistema tiene derecho a conocer qué se va a hacer con esa información que haya proporcionado y para qué va a servir.

La temática del juego está ambientada en el folklore europeo, más específicamente del Este de Europa, haciendo alusiones a criaturas fantásticas, tradiciones y mitología. Además, extrae determinados conceptos de los libros de fantasía de *Andrzej Sapkowski* con su famosa saga de *Geralt De Rivia*. Al contemplar conceptos de esta saga, es necesario aludir a los Derechos de Propiedad Intelectual. Dentro del apartado de Derechos de Autor, también conocido como *copyright*, hay una sección dedicada al ámbito de los videojuegos donde declara que, para determinados países del mundo, incluido España, se debe aprobar un examen de legislación que evalúa el propio autor pudiendo éste impartir ciertas cesiones [41]. En resumidas cuentas, los autores marcan los ámbitos más beneficiosos sobre la situación jurídica que afecta tanto a dichos autores como a los videojuegos de forma enlazada.

La incorporación de estos conceptos en el proyecto no produce ningún tipo de beneficio para el creador o este proyecto que pueda afectar al derecho de remuneración del autor y, además, no perjudica injustificadamente los intereses del autor. Al cumplirse estos aspectos y mantener al proyecto en un entorno privado sin exponerlo al público, el proyecto se mantiene dentro del marco legal de la Propiedad Intelectual.

7. ENTORNO SOCIO-ECONÓMICO

En este apartado del documento se analiza el impacto socio-económico del proyecto realizado y se calcula el presupuesto necesario para llevarlo a cabo.

Además de los impactos que pueda tener este proyecto, se analiza el presupuesto alcanzado para la venta de éste. Se hará un desglose de cada una de las fuentes de gastos y un estudio del coste final.

7.1. IMPACTO SOCIO-ECONÓMICO

Todo proyecto enfocado a un uso público o que vaya a ser utilizado por personas ajenas, requiere un estudio de su impacto en ese público. No sólo se debe prestar atención a aspectos económicos o sociales sino, además, a impactos medioambientales, éticos…

Por ello, se requiere hacer una separación de cada uno de los distintos tipos de impactos que puede tener un proyecto en el mundo real. El impacto positivo en uno de los tipos no implica necesariamente que sea positivo en los otros, de hecho, el dilema aparece cuando se da este caso. En esta situación, la responsabilidad cae sobre los analistas que revisan la viabilidad del proyecto en función de estos impactos y, en función de los resultados, se deben realizar cambios en determinados aspectos del proyecto.

Como **impacto social** se debe revisar qué repercusión tiene sobre las personas. Se debe estimar si el proyecto tiene una utilidad para la sociedad o algún propósito para las personas que lo vayan a utilizar. En el caso de este proyecto, no presenta ningún tipo de utilidad real dentro de la sociedad, aunque su propósito es exclusivamente el entretenimiento.

A la hora de estudiar el **impacto económico** de un proyecto se debe centrar la atención en la rentabilidad de su desarrollo, es decir, conocer si el proyecto va a proporcionar beneficios para sus creadores. En el caso del proyecto actual, no se puede hablar de un beneficio económico ya que entraría en conflicto con lo tratado en el marco legal refiriéndose a la Propiedad Intelectual. Sin embargo, se podría hablar de ventas y beneficios, si la temática y los elementos utilizados de las novelas se sustituyesen por otros de carácter propio.

Uno de los impactos más amplios y conflictivos es el que trata el **impacto ético,** donde se estudia cómo puede afectar un proyecto a las personas. A diferencia del impacto social, el impacto ético se centra en cómo afecta el proyecto de forma negativa a las personas. En el mundo de los videojuegos este tipo ha crecido en importancia a lo largo de los años. Actualmente, hay una estricta política que controla que los videojuegos no ataquen a la dignidad de ninguna persona o colectivo y que no atenten contra minorías o promuevan delitos de odio. En el caso de este proyecto, el impacto ético es positivo ya que no afecta de forma negativa ni a colectivos ni particulares.

Al tratarse de un proyecto de *software* el **impacto medioambiental** no tiene una presencia tan fuerte como los otros anteriores. Este impacto analiza la repercusión en el medio ambiente que puede ocasionar la realización de un proyecto. Sin embargo, tanto su

desarrollo como su implantación en la sociedad no afecta de ninguna forma al medio ambiente.

7.2. PRESUPUESTO

Para poder vender el proyecto, se debe elegir uno de los dos caminos disponibles en función del conflicto establecido en el marco legal de la Propiedad Intelectual. Una opción sería realizar un pago para los derechos de autor y la otra opción sería cambian la temática del juego para no entrar en conflicto con los derechos de autor.

La opción elegida es la segunda: eliminar los elementos provenientes de las novelas y reemplazarlos por aspectos de creación propia. Esto supone un descenso en el coste del proyecto.

Dentro del presupuesto del proyecto podemos separar los costes en dos grupos: costes de conexión directa al proyecto y costes indirectos. Estos primeros se refieren a la inversión necesaria para la realización de las tareas del desarrollo mientras que los segundos costes se refieren a conceptos consumibles como electricidad o Internet. Para los costes directos se entrará en detalle haciendo un desglose mientras que, para los indirectos se proporciona un valor conjunto.

Dentro de los costes directos, se incluye a todo el personal involucrado en la creación del proyecto. Para el personal se necesita saber el coste a la hora y el coste total. Dentro del personal tenemos: un **analista** que cumple las tareas de análisis y diseño, un **desarrollador** que se encarga de traducir todo a código y de efectuar las pruebas de sistema y, finalmente, el **jefe de proyecto** que lleva toda la administración.

Personal	***Coste por hora (€/hora)***	***Coste total (€)***
Analista	20	2200
Desarrollador	15	2850
Jefe de Proyecto	25	2250
	Total:	7300

Tabla 69: Presupuesto de personal

Por otro lado, en los costes directos, tenemos el gasto en equipamiento y *software*. Aquí se incluyen los ordenadores utilizados. Sin embargo, se debe trabajar con el coste amortizado a partir de la fórmula:

$$Coste\ amortizado = \frac{Meses\ del\ proyecto}{Meses\ útiles\ del\ equipo} * Coste\ del\ equipo$$

Al tratarse de un proyecto de 12 meses y cada equipo tener unos 4 años útiles (48 meses), obtenemos el valor asociado de coste amortizado.

Elemento	***Coste (€)***	**Coste amortizado (€)**
Ordenador sobremesa (i7 8GB RAM)	900	225
Portátil (i5 4GB RAM)	500	125
	Total:	350

Tabla 70: Presupuesto de equipamiento

Finalmente, juntamos ambas partes de los costes directos con el valor total de los costes indirectos. Estos costes indirectos suponen un 20% de los costes de personal.

Tipo de coste	***Coste (€)***
Personal	7300
Equipamiento	350
Costes indirectos (20% de costes personales)	1460
Total:	9110

Tabla 71: Presupuesto total

Sin embargo, se debe añadir un 15% adicional como gastos adicionales de riesgo. Es decir, en caso de gastos imprevistos o en caso de accidentes. Pero si se quiere tener un beneficio, se debe volver a aumentar el gasto total. Este aumento es de 15% también.

Por tanto, si el gasto total sin aplicar los aumentos es 9,110 €. Entonces, al aplicar un total de 30% de coste adicional, se quedaría un gasto final de 11,843 €.

7.3. PLANIFICACIÓN

La planificación del proyecto empieza antes de verano, cubre todos los meses de la estación y acaba al mismo tiempo. La planificación sigue una estructura que se divide en varias partes, que generalmente coinciden con las secciones del documento.

Para mostrar la evolución de cada una de las partes, se utiliza un Diagrama de Gantt, como se muestra en la figura 28, donde para cada parte se marca la fecha de inicio y la fecha final. Además de cada parte del proyecto, se muestra el tiempo empleado para el proyecto en su totalidad.

Las tareas se dividen en memoria, estado del arte, análisis, diseño, implementación y pruebas.

La memoria o documentación ocupará todo el tiempo del proyecto ya que se hace a medida que se avanza por las distintas partes. Por ello, empieza el 15 de marzo y acaba el 31 de agosto.

El estado del arte ocupa la primera tarea para poder saber cómo se va a enfocar el proyecto y qué se va a utilizar. Esta tarea empieza el 15 de marzo y acaba el 10 de abril.

El análisis, que cubre todos los requisitos y la batería de pruebas, ocupa algo más que el estado del arte, empezando según acabe la sección anterior, el 10 de abril, y acabando el 17 de mayo.

La sección de diseño requiere algo más de tiempo que el análisis ya que se necesita una construcción de una arquitectura, componentes y la exposición de gran cantidad de diagramas. Por ello, empieza el 17 de mayo y acaba el 27 de junio.

La implementación ocupa aún más tiempo que el diseño ya que no solo hay que construir un sistema inteligente, sino que se necesita desarrollar el juego en el que va a estar incluido. Empieza el 27 de junio y acaba el 15 de agosto.

Finalmente, la sección de pruebas lleva menos tiempo que cualquier otra sección ya que la batería de pruebas ya ha sido establecida en el análisis y solo falta su ejecución. Por ello empieza el 15 de agosto y acaba el 31 de agosto.

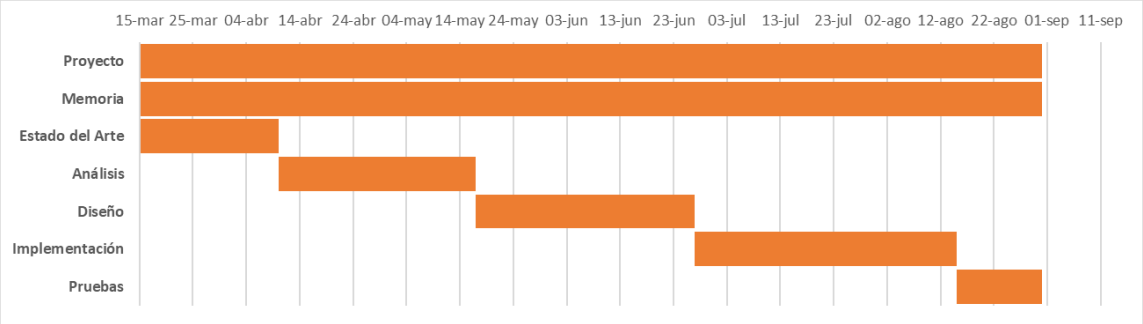

Figura 28: Diagrama de Gantt de planificación del proyecto

8. CONCLUSIÓN

Al encontrarse en un momento en la historia donde tanto el mundo de los videojuegos como el mundo de la Inteligencia Artificial están en su etapa avanzada de crecimiento, cada experto, aficionado y persona interesada tiene su atención puesta sobre ellos. Este hecho, junto con el éxito obtenido, va a facilitar e incentivar aún más el crecimiento de ambos campos en los próximos años. Este proyecto nace gracias a la motivación generada por el crecimiento de ambas en tan poco tiempo.

El objetivo de este proyecto está dividido en dos partes que a su vez están conectadas: una primera parte que estudia si el uso de Aprendizaje por Refuerzo para un juego por turnos da resultados notables y una segunda parte que se dedica a la creación del sistema de un videojuego que incluya un *software* capaz de realizar las tareas de la primera y dé un propósito a la IA implementada.

A su vez, la primera parte, en su desarrollo, tiene un objetivo secundario pero relevante para el proyecto: que el agente implementado por el aprendizaje no sea demasiado difícil de vencer pero que, a su vez, no sea excesivamente fácil, es decir, balancear el nivel de dificultad del agente.

Habiendo establecido ambos objetivos, la evolución del proyecto va encaminado a su consecución. Al marcar cada una de las partes en un planteamiento puramente teórico, el enfoque y las funcionalidades sufrieron ligeros cambios cuando se llevó a cabo su implementación. Por ello, se han elaborado distintas pruebas, procedimientos y estudios de las diferentes alternativas para ejercer cada tarea.

Un ejemplo claro de este fenómeno es el balanceo del combate. El entrenamiento del agente automático no es el único factor que influye en los enfrentamientos entre el agente y los jugadores; hay otros aspectos que pueden inclinar la balanza hacia un lado u otro. Y aquí recalcamos el objetivo secundario del primer objetivo del proyecto: crear un agente que suponga un desafío real para el jugador. Sin embargo, aquí están implícitos los aspectos que desnivelan la balanza. Cuando se estructuró el sistema de combate con todos los valores, fórmulas y elementos asociados, se obtuvieron resultados muy desfavorables para el agente automático, no porque su entrenamiento fuese incompleto o porque los parámetros del algoritmo fuesen los incorrectos, sino porque la estructuración de los valores de daño y de los atributos estaban favoreciendo de forma desmesurada a los jugadores. Después de unos cambios en los parámetros del combate, sin modificar el tipo de entrenamiento ni parámetros del algoritmo, se obtuvieron unos resultados mucho más equilibrados.

El primer objetivo del proyecto se alcanzó gracias a procedimientos como la construcción de una extensa batería de pruebas. Sin embargo, para poder elaborar una batería de pruebas que fuese coherente y evitara redundancias era necesario realizar un estudio previo del algoritmo que se iba a emplear. Esto facilitó una mayor atención a pruebas con un valor de un parámetro dentro de un rango y a la comprensión de los resultados obtenidos. En el caso del algoritmo de este proyecto, gracias al estudio previo en el Estado del Arte, se supo qué implicaba el uso de un valor bajo de Gamma y qué evolución debían tener los parámetros de Alfa y Épsilon. Sin este estudio previo, se hubieran realizado el

doble de las pruebas establecidas en este documento, aunque la mayoría carecerían de sentido si no se sigue un procedimiento lógico.

La construcción del Motor del juego junto con sus elementos asociados, incluido el motor de aprendizaje, marcan el alcance del segundo objetivo. Al igual que el primer objetivo, esta parte también requirió un estudio previo de determinados aspectos que determinaran el camino a seguir en la construcción del sistema. Aunque no se requieran pruebas para calcular el mejor rendimiento de un elemento, como el agente automático, se necesitan pruebas, como se ha mencionado antes, para encontrar un equilibrio en el sistema de combate. No solo eso, sino que además para poder cumplir realmente el objetivo se establecieron unas pruebas que regulasen el buen funcionamiento de dicho sistema. La implementación de un sistema como el del proyecto tiene una serie de procedimientos previos que aseguran el correcto funcionamiento de éste en etapas futuras.

Este proyecto no solo alcanzó los objetivos planteados, sino que también demostró que la construcción de un sistema de juego y un agente automático no supone un trabajo excesivamente complicado. Además, consiguió estandarizar ambos conceptos de videojuegos e Inteligencia Artificial que están tan presentes hoy en día.

Por último, muchos elementos adicionales pueden ser añadidos al proyecto y son estos detalles los que proporcionarían un mayor entretenimiento del juego. Sin embargo, ese aspecto no es el objetivo principal del mismo. Estos elementos adicionales se mencionan en el próximo apartado.

9. FUTUROS PROYECTOS

A partir del sistema obtenido en este proyecto, muchas mejoras y añadidos existen al alcance de uno. Al tratarse de un juego, la cantidad de distintas funcionalidades, mecánicas y elementos a incluir es inmensa, pero hay ciertos elementos que constituyen la jugabilidad de muchos otros juegos que se podrían añadir a éste.

Como se ha mencionado previamente, se busca que el agente automático sea un reto para todos los arquetipos de jugadores que puedan experimentarlo. Pero esa tarea es difícil para un mismo tipo de agente y puede ocasionar que alguno de los jugadores encuentre al agente demasiado fácil o demasiado difícil. Por ello, muchos juegos presentan la existencia de un selector de dificultad. Esto no solo permite aumentar las funcionalidades del sistema, sino que además permite al programador insertar otro tipo de agente para los nuevos niveles de dificultad. No tiene por qué ser otro tipo de agente, sino que puede tratarse de un agente de Aprendizaje por refuerzo pero que tenga otro tipo de estado, otros parámetros o que simplemente tenga más información del entorno.

Otros elementos que tienen posibilidad de ser añadidos a la jugabilidad pueden ser más clases que se diferencien no solo por los rasgos de este proyecto sino por sus habilidades o por su propósito en la partida; la implementación de habilidades especiales; más inventario para los personajes; algún evento especial que rompa la monotonía de las mecánicas.

Quizá el rasgo que más hace falta en este proyecto es la implementación de un entorno gráfico. Supondría una mejora ya que los jugadores seguirían mejor el avance de la partida y tendría una mayor inmersión en la historia.

REFERENCIAS

[1] C. Birch, «Understanding Pac-Man Ghost Behaviour,» *http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior,* vol. 2010, 2 Diciembre 2010.

[2] D. Winter, «Historia del Pong (inglés),» *http://www.pong-story.com/intro.htm,* 5 Junio 2013.

[3] «Atari Webpage».*https://www.atari.com/.*

[4] «Age Of Empires WebPage».*https://www.ageofempires.com/.*

[5] «Pokemon types & type chart».*https://pokemondb.net/type.*

[6] P. Cardwell, «Skeptical inquirer,» *Vol 18,* Winter 1994.

[7] A. Mullins, «Murderers remain a mystery: Shocked community stymied by Dungeons and Dragons style murders,» *Vancouver Sun,* 5 Junio 1992.

[8] T. Wilson, J. Clark y K. Kershner, «Beginner's Guide to Dungeons & Dragons».

[9] «Reglas y documentación de Dungeons & Dragons».*https://www.dndbeyond.com/compendium/rules.*

[10] «Razas de Dungeons & Dragons».*https://www.dndbeyond.com/characters/races.*

[11] «Clases de Dungeons & Dragons».*https://www.dndbeyond.com/characters/classes.*

[12] «Equipamiento de Dungeons & Dragons».*https://www.dndbeyond.com/equipment.*

[13] «Objetos de Dungeons & Dragons».*https://www.dndbeyond.com/magic-items.*

[14] E. Even-Dar y Y. Mansour, «Learning Rates for Q-learning,» *Journal of Machine Learning Research,* 12 Marzo 2003.

[15] M. Tokic, «Adaptive ε-greedy exploration in reinforcement learning based on value differences,» *Annual Conference on Artificial Intelligence,* 2010.

[16] J. I. Bagnato, «Qué es overfitting y underfitting y cómo solucionarlo,» *AprendeMachineLearning,* 12 Diciembre 2017.

[17] S. Whiteson, B. Tanner, M. Taylor y P. Stone, «Protecting against evaluation overfitting in empirical reinforcement learning,» 2011.

[18] Z.-L. He y P.-K. Wong, «Exploration vs. Exploitation: An Empirical Test of,» Julio-Agosto 2004.

[19] L. Buchanan, «Top 10 Classic Shoot 'Em Ups,» *IGN,* 26 Mayo 2009.

[20] «Battle Garegga».*https://segaretro.org/Battle_Garegga.*

[21] M. Čertický y D. Churchill, «The Current State of StarCraft AI Competitions and Bots.».*In Proceedings of the AIIDE 2017 Workshop on Artificial Intelligence for Strategy Games. 2017.*.

[22] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontanón y M. Čertický, «StarCraft Bots and Competitions.».*Chapter in Encyclopedia of Computer Graphics and Games (ECGG). Springer International Publishing. ISBN: 978-3-319-08234-9. 2016.*.

[23] I. Viana, «Deep Blue, la máquina que desafió la inteligencia humana,» *ABC,* 10 Febrero 2010.

[24] D. Silver, «Mastering the game of Go with deep neural networks and tree search,» *International Journal of Science (www.nature.com),* 2016.

[25] J. Heinrich y D. Silver, «Deep reinforcement learning from self-play in imperfect-information games,» *NIPS Deep Reinforcement Leaning Workshop,* 2016.

[26] «Página oficial de TCEC».*http://tcec.chessdom.com/.*

[27] M. Delarche, «La victoire de Stockfish au championnat du monde des ordinateurs,» *MediaPart,* 5 Julio 2018.

[28] D. Silver , T. Hubert y J. Schrittwieser, «Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,» *Cornwell University Library, Artificial Intelligence,* 5 Diciembre 2017.

[29] J. Vincent, « DeepMind’s AI became a superhuman chess player in a few hours, just for fun,» *The Verge,* 6 Diciembre 2017.

[30] S. Knapton, «Entire human chess knowledge learned and surpassed by DeepMind's AlphaZero in four hours,» *The Telegraph UK,* 6 Diciembre 2017.

[31] Krusher, «Tetris: Versiones, historia y curiosidades,» *PixFans* , 3 Febrero 2009.

[32] M. Machuca, «Mancala. Historia y reglas del milenario juego de mesa,» *AboutEspañol,* 2 Noviembre 2016.

[33] «Mancala - Reglamentos de tablero».*http://www.acanomas.com/Reglamentos-Juegos-de-Tablero/085/Mancala.htm.*

[34] E. Frank, M. Hall y I. Witten, «The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"».*Morgan Kaufmann, Fourth Edition, 2016.*

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann y I. Witten, «The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1.».

[36] J. Reutemann, P. Pfahringer y B. Holmes, «MEKA: A Multi-label/Multi-target Extension to Weka,» *Journal of Machine Learning Research*, 2016.

[37] T. E. o. E. Britannica, «Ghoul».*https://www.britannica.com/topic/ghoul.*

[38] P. Geller, «Banshee».*https://mythology.net/mythical-creatures/banshee/.*

[39] Boletin Oficial del Estado, «Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal.».

[40] «Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo,» 27 Abril 2016.

[41] «OMPI (Organización Mundial de la Propiedad Intelectual) - Copyright en Videojuegos».*http://www.wipo.int/copyright/es/activities/video_games.html.*

ANEXO

A. GLOSARIO

NPC: *Non Playable Character*. Se dice de los elementos de los juegos que representan a seres vivos pero que el jugador no puede manejar. Pueden tratarse de aliados, neutrales o enemigos.

Simulador: Agente capaz de emular las acciones y comportamientos de un humano mediante el estudio de partidas de éste.

Agente: En aprendizaje automático, sigue el modelo fijado en dicho aprendizaje.

Modelo: Políticas, comportamientos y mecánicas que establecidas después del aprendizaje de una máquina.

Python: Lenguaje de programación multiparadigma que permite la orientación a objetos, programación imperativa y programación funcional.

Learning: Palabra inglesa para referirse al 'Aprendizaje', el proceso de mejora para realizar una tarea determinada con el fin de poder actuar de forma autónoma más adelante.

Estado-acción: En Aprendizaje por Refuerzo, es una tupla asociada al agente que representa, respectivamente, la situación o las condiciones en las que se encuentra y la acción tomada por el agente según esa situación.

Jugabilidad: Característica de un juego que marca la buena o mala experiencia que ha tenido un usuario según el manejo y desarrollo de dicho juego.

Clasificador: Algoritmo que asigna una instancia de entrada sin etiquetar en una categoría conocida determinada.

Instancia: Un dato del problema que se quiere resolver.

B. MAPA DE ‘EL CONTINENTE’

Mapa a partir del cual se extraen los nodos de movimiento en la parte de exploración del juego. Extraído de la primera página del quinto libro de la saga.